Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Tietokantajärjestelmät

2012

Mikko Kankaanranta

HUUTOKAUPPA-JÄRJESTELMÄN TOTEUTUS CODEIGNITER-OHJELMOINTIKEHYKSESSÄ

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely | Tietokantajärjestelmät

Kesäkuu 2012 | Sivumäärä: 66

Ohjaaja Päivi Nygren

Mikko Kankaanranta

HUUTOKAUPPAJÄRJESTELMÄN TOTEUTUS CODEIGNITER-OHJELMOINTIKEHYKSESSÄ

Tässä opinnäytetyössä dokumentoidussa projektissa rakennettiin sovellus, jonka avulla voidaan perustaa web-huutokauppa. Sovellus ohjelmoitiin CodeIgniter-ohjelmointikehyksessä. Projektin lähtökohtana oli ajatus helposti asennettavasta ja käyttöönotettavasta huutokauppa-alustasta, joka sulkisi huutokaupalle ominaisen monimutkaisen logiikan yksinkertaisen, koodin tasolla olevan käyttöliittymän alle.

Tavoitteena oli tuottaa lopputulos joka antaa huutokaupan perustajalle ohjelmointikehyksen mukana tulevat www-sovelluskehitykseen suunnatut työkalut ja alustan ratkaisut huutokaupan perustamiseen, ylläpitämiseen ja kehitykseen liittyvissä toimissa samassa kokonaisuudessa.

Huutokauppa-alusta ohjelmoitiin php-pohjaisessa CodeIgniter-ohjelmointikehyksessä, jossa tietokantaratkaisuna käytettiin MySQL tietokantaa. Alustan toteutuksessa otettiin huomioon erilaiset hosting-ympäristöt, jota varten tietokannasta poistettiin tuki taulujen välisille yhteyksille. Viite-eheydestä huolehditaan ohjelmakoodissa. Koko huutokauppa-alusta ohjelmointiin noudattaen CodeIgniter-ohjelmointikehyksen implementoimaa MVC (mode-view-controller) sovellusarkkitehtuuria.

ASIASANAT:

Web-huutokauppa, CodeIgniter, ohjelmointikehys, php

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Programme: Business Information Technology | Database Systems

June 2012 | Total number of pages: 66

Instructor Päivi Nygren

Mikko Kankaanranta

IMPLEMENTATION OF AUCTION SYSTEM IN CODEIGNITER FRAMEWORK

This thesis presents an application that was built for establishing web-based auctions. The application was programmed in the CodeIgniter framework. The incentive for the project was an idea of an auction system that conceals all complicated logic related to auctions under simple code interface and would be is easy to deploy and use.

The objective was to produce an application that gives all the tools of the framework and solutions implemented by the auction system in the hands of the founder of auctions in order to develop and maintain the application.

The auction system was programmed in the php-based application development framework called CodeIgniter, which used MySQL as database solution. Different hosting environments were taken into consideration when implementing the system, especially in the design of the database. Support for referential integrity in the database was not implemented. Instead, it is taken care of in the code. The auction system was programmed following the MVC (model-view-controller) pattern implemented by the CodeIgniter framework.

KEYWORDS:

Web-auction, CodeIgniter, framework, php

SISÄLTÖ

LIITTEET

KUVAT

KUVIOT

SANASTO

MVC	Model-View-Controller

1 JOHDANTO

Ohjelmointikehys termi on syntynyt tarpeesta tuottaa sovelluksia nopeammin. Sen tarkoitus on helpottaa ja nopeuttaa kehittäjien työtä tarjoamalla valmiita työkaluja yleisten toimenpiteiden suorittamiseen ja ongelman ratkontaan, jotta kehittäjät voisivat keskittyä enemmän siihen miten sovellus toimii ja miltä se näyttää. Ohjelmointikehys terminä on teknologiariippumaton ja sen implementoima ajatusmalli on sovellettavissa riippumattomasti ison tai pienen sovelluksen tuottamiseen. Ohjelmointikehyksen ajatus voidaan myös viedä pidemmälle. Sen sijaan että sillä tuotettaisiin sovellus, sitä voidaan käyttää pohjana alustalle, joka hieman sovelluksesta poiketen pyrkii ratkaisemaan jonkin tietyn, hyvin spesifisen ongelman. Tällä tavalla pystytään tuottamaan ratkaisu, jossa on mukana ohjelmointikehyksen työkalut ja alustan ratkaisut tiettyyn, spesifiseen ongelmaan.

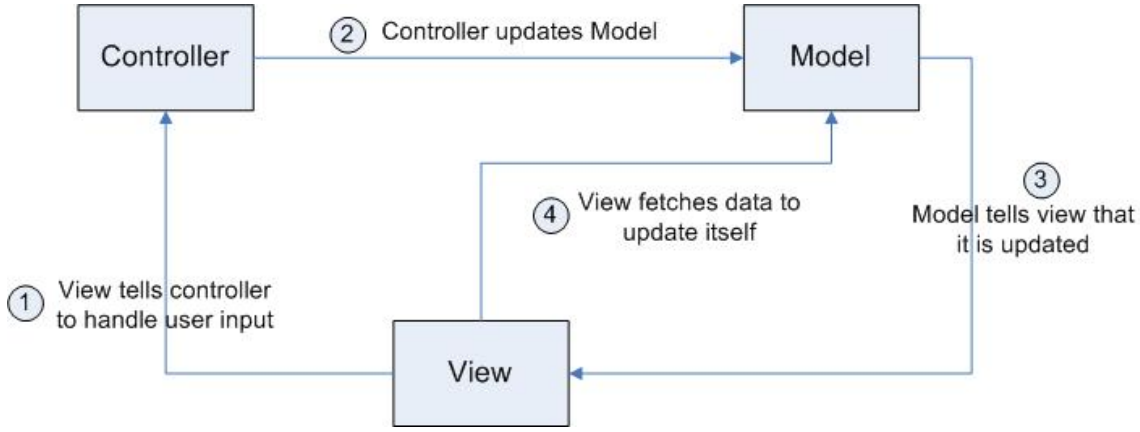
Verkossa toimiva huutokauppa on tuonut kaupankäynnin massoille. Kuitenkin huutokaupan monimutkainen logiikka on rajoittanut niiden määrää merkittävästi. Entä jos olisi olemassa huutokauppa-alusta joka olisi helppo asentaa ja käyttöönottaa. Siinä olisi joustava ja helposti muokattava käyttöliittymä ja paljon työkaluja, jotka pyrkivät ratkaisemaan huutokauppaan liittyvät ongelmat samalla tavalla kuin ohjelmointikehys ratkaisee sovelluksen tuottamiseen liittyviä ongelmia. Se olisi rakennettu ohjelmointikehyksen päälle tuoden lisää työkaluja itse www-sovelluskehitykseen. Tähän ongelmaa tämä opinnäytetyö pyrkii vastaamaan.

2 CODEIGNITER-OHJELMOINTIKEHYS JA HUUTOKAUPPA

2.1 CodeIgniter

Ohjelmointikehys (framework) on suunniteltu tukemaan ja nopeuttamaan kehitystyötä tarjoamalla valmiita kirjastoja sovelluskehityksessä tarvittaviin yleisiin toimenpiteisiin. CodeIgniter on verkkosovellusten rakentamiseen suunniteltu, PHP:n perustuva oliopohjainen ohjelmointikehys (web application framework) jonka kehityksestä vastaa EllisLab niminen yritys. CodeIgniterin tehtävä on nopeuttaa kehittäjän työtä Web sovelluksia rakennettaessa tarjoamalla valmiita kirjastoja mm. lomakkeisiin, reititykseen, turvallisuuteen, sessioihin ja tietokantoihin liittyvissä tehtävissä. CodeIgniter tukee modulaarista suunnittelumallia jossa se ei itsenäisesti lataa kehyksen resursseja, vaan kehittäjä itse määrittää omien tarpeidensa mukaan, mitä resursseja haluaa käyttää. Jos ladattavalla näkymällä ei tarvita yhtään lomaketta, ei silloin tarvita myöskään siihen suunnattua resurssia.

CodeIgniter perustuu MVC-arkkitehtuuriin (model-view-controller) joka on yksi ohjelmistoarkkitehtuurityyli (Wikipedia 2012, MVC-arkkitehtuuri [viitattu 6.6.2012]). MVC-malli eristää käyttöliittymän sovelluksen logiikasta. MVC:ssä model (malli) käsittelee sovelluksen tietomallia ja sisältää toiminnallisuuden, joka mahdollistaa vuorovaikutuksen tietokannan kanssa, esimerkiksi tietueen lisäyksen tai poiston. View (näkymä) käsittelee käyttöliittymää ja siinä näytettävää tietoa. Controller (käsittelijä) elää näkymän ja mallin välissä, huolehtien sovelluksen raskaasta logiikasta ja tiedosta joka liikkuu mallin ja näkymän välillä.

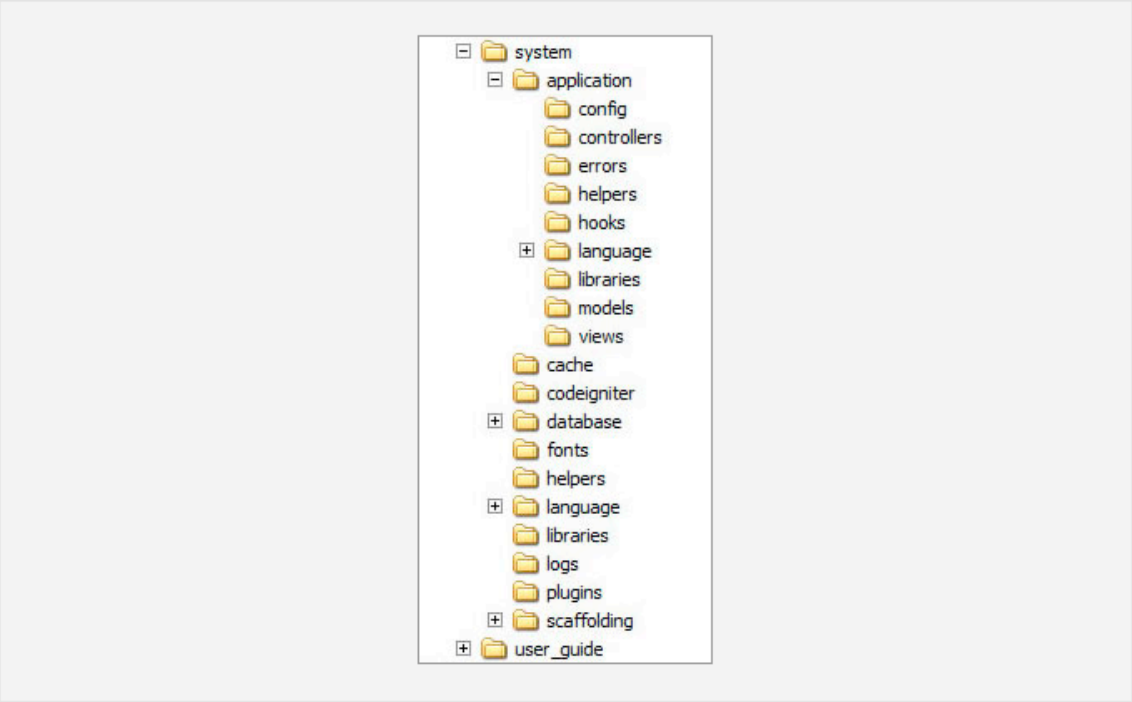


Kuvio 1. Model-View-Controller ohjelmistoarkkitehtuuri [viitattu 6.6.2012]. Saatavissa: http://www.actionscript.org/resources/articles/935/1/Cairngorm-Getting-Started--Part-1/Page1.html .

Kuvion 1 kohdassa 1 näkymä (view) käskee kontrollerin (controller) käsitellä käyttäjän syötteen. Kohdassa 2 käsittelijä päivittää sovelluksen mallin (model). Kohdassa 3 malli kertoo näkymälle, että malli on päivitetty. Kohdassa 4 malli syöttää tiedon näkymälle jotta se voi päivittää itsensä.

2.2 CodeIgniterin hakemistorakenne

CodeIgniter sisältää omanlaisensa hakemistorakenteen jossa on pyritty erottelemaan toisistaan eroavat toiminnallisuudet omiin hakemistoihinsa ja ohjelmointikehyksen ydin on selvästi erillään kehittäjän hakemistoista. Jokaisella hakemistolla on oma tarkoituksensa ja ne yleensä sisältävät CodeIgniterin mukaan nimettyjä tiedostoja, jotka ovat joko luokkia, konfigurointi-tiedostoja tai tiedostoja jotka, sisältävä funktioita.



Kuva 1. CodeIgniterin hakemistorakenne [viitattu 6.6.2012]. Saatavissa: http://codeignitertutorials.blogspot.fi/2009/05/codeigniter-installation-step-2_10.html.

Seuraavassa kuvataan hakemistojen tarkoitus. Tasojen ylimpänä oleva system-hakemisto, joka sisältää kaiken Codeinginterin toiminnallisuuden. System-hakemiston alihakemistoihin voi myös luoda omia kirjastoja (library) ja auttaja-tiedostoja (helper), mikäli kehittäjä haluaa luoda toiminnallisuutta, joka on saatavilla läpi kehyksen ja sen kaikkien sovellusten. Tämä on loogista silloin, kun kehittäjä haluaa luoda monta sovellusta samalle asennukselle. ./application-hakemiston ei ole pakko olla samanniminen, kehittäjä voi nimetä sen halutessaan uudelleen.

./application-hakemisto on rakenteeltaan melkein identtinen ylätason system-hakemiston kanssa. Tällä tavalla kehittäjällä voi luoda kirjastoja (library), auttajatiedostoja (helper), käsittelijöitä (controller), malleja (model) ja näkymiä (view) jotka ovat yksilöllisiä vain luotavalle sovellukselle eikä koko kehykselle. Kehittäjä luo sovellukseen toiminnallisuutta irrallaan kehyksestä ja sen toiminnallisuudesta. Kaikki tämän hakemiston alla oleva toiminnallisuus on sovellukselle yksilöllistä ja kehittäjän luomaa.

./application/config-hakemisto sisältää kaikki kaikki konfigurointitiedostot, jotka ovat relevantteja kehittäjän luomalle sovellukselle. Nämä tiedostot sisältävät tietoa mm. tietokannan yksityiskohdista ja mitä kirjastoja CodeIgniter lataa automaattisesti.

./application/controllers-hakemistoon kehittäjä luo sovellukselle yksilölliset käsittelijät (controller) CodeIgniterissa käsittelijä on luokka, joka on sijoitettu ./application/controllers-hakemistoon ja nimetty Codeignetrin ohjeiden mukaisesti. Esimerkki alla kuvaa tätä.

```
/**
./application/controllers/samplecontrolle.php
*/
<?php
class Samplecontroller extends CI_Controller {

    public function index()
    {

    }
}
?>

</ end of sample code>
```

./application/errors-hakemisto sisältää kaiki sovelluksen virhesivujen näkyät (view).

./application/helpers-hakemisto sisältää kaikki sovellukselle yksilölliset auttajatiedostot (helper). Auttajatiedosto sisältävät php-funktioita jotka on tarkoitettu helpottamaan pienten asioiden tekemistä. Auttajatiedosto ladataan vain tarvittaessa. Hyvä esimerkki on CodeIgniterin html auttaja (./system/helpers/html_helper.php) jossa on funktio br. Esimerkki alla kuvaa tätä.

```
/**
./application/helpers/html_helper.php
*/
// ------------------------------------------------------------------
/**
* Generates HTML BR tags based on number supplied
*
* @access       public
```

```
* @param       integer
* @return      string
*/
if ( ! function_exists('br'))
{
  function br($num = 1)
  {
     return str_repeat("<br />", $num);
  }
}

</ end of sample code>
```

./application/hooks-hakemistoon voi luoda tiedostoja muokkaamaan CodeIgniterin ytimen toiminnallisuutta. CodeIgniterin virtaus on jaettu vaiheisiin, joihin on mahdollista kiinnittyä koukkujen (hook) avulla. Kehittäjä voi määrittää hakemistossa ./application/config/config.php tiedostoon polun, tiedostonimen, luokan, metodin ja parametreja jotka suoritetaan ennalta määritetyissä vaiheissa CodeIgniterin virtausta. Esimerkki alla kuvaa tätä.

```
<?php
$hook['pre_controller'] = array(
  'class'      => 'MyClass',
  'function' => 'Myfunction',
  'filename' => 'Myclass.php',
  'filepath' => 'hooks',
  'params'   => array('beer', 'wine', 'snacks')
);

</ end of sample code>
```

”pre_controller”-hook suoritetaan heti, kun käyttäjän luoman sovelluksen käsittelijät (controller) on ladattu. Ennen kuin koukkuun määritetyt toimet suoritetaan, CodeIgniterin ytimen luokat on ladattu, reititys ja turvallisuus tarkastukset on suoritettu.

./application/language-hakemistoon voi tallentaa tekstejä, jotka ladataan language-kirjastoon määritettyjen avaimien avulla, kun luodaan monikielistä sovellusta. Esimerkiksi virheilmoitukset on mahdollista kieliversioida tämän hakemiston kautta. Esimerkki alla kuvaa tätä.

```
/**
./application/languages/en_lang.php
*/
$lang['error_email_missing'] = "You must submit an email address";

</ end of sample code>
```

./application/libraries-hakemistoon kehittäjä voi luoda omia kirjastoja (library), laajentaa CodeIgniterin olemassa olevia kirjastoja tai ylikirjoittaa niitä kokonaan. Kirjastolla viitataan luokkaan joka on sijoitettu ./application/libraries-hakemistoon tai ./system/libraries-hakemistoon ja on nimetty CodeIgniterin ohjeiden mukaisesti alkamaan isolla kirjaimella: ./application/libraries/Someclass.php. Esimerkki alla kuvaa tätä.

```
<?php if ( ! defined('BASEPATH')) exit('No direct script access al-
lowed');

/**
./application/libraries/Someclass.php
*/
class Someclass {

  public function some_function()
  {

  }
}
/* End of file Someclass.php */
</ end of sample code>
```

./application/models-hakemistoon kehittäjä voi luoda malleja (model), jotka ovat sovelluksen CodeIgniterin arkkitehtuuria noudattava rajapinta tietokantaan. Mallilla, samalla tavalla kuin kirjastolla, tarkoitetaan luokkaa, joka on sijoitettu application/models-hakemistoon ja joka noudattaa CodeIgniterin määrittämää nimeämismallia. Tiedoston nimi pitää noudattaa formaattia kehittajanmallinnimi_model.php. Jokainen CodeIgniterin malli perii CodeIgniterin CI_Model nimisen luokan. Esimerkki alla kuvaa tätä.

```
<?php

/**
./application/models/kehittajanmallinnimi_model.php
*/

class kehittajanmallinnimi_model extends CI_Model {

  function __construct()
  {
    parent::__construct();
  }

}

</ end of sample code>
```

./application/views-hakemistoon kehittäjä luo sovelluksen teematiedostot joita myös kutsutaan template-nimellä. Sovelluksen kontrollerit käsittelevät tietoa ja luovuttavat sen lataamilleen views-hakemiston teematiedostoille, jotka piirtävät lopullisen näkymän käyttäjän ruudulle. Teematiedosto on PHP-tiedosto, jossa on sekaisin PHP:ta ja HTML:ää. CodeIgniterissa ei ole ennalta määrätty nimeämispolitiikka views-hakemiston tiedostoille, koska käsittelijät lataavat views-hakemiston tiedostot kehittäjän määritysten mukaan. Alla oleva esimerkki kuvaa, tilannetta jossa kehittäjä haluaa ladata teematiedoston ./application/views/minaolenview.php.

```
<?php

/**
application/controllers/samplecontroller.php
*/

/**
Samplecontroller-niminen controlleri lataa template tiedoston mi-
naolenview.php:
*/

class Samplecontroller extends CI_Controller {

  public function index()
  {
     $this->load->view('minaolenview');
  }
```

```
}

?>

</ end of sample code>
```

./system/cache-hakemisto pitää sisällään kaikki caching-kirjaston välimuistia varten generoidut tiedot.

./system/codeigniter-hakemistossa on kaikki toiminallisuus mikä saa CodeIgniterin toimaan. Kutsutaan myös coreksi, eli ytimeksi.

./system/database-hakemistossa on tietokantarajapintaa varten ajurit ja luokat, joiden avulla CodeIgniter voi käyttää tietokantaa riippumatta käytettävästä tietokantateknologiasta.

./system/fonts-hakemistossa sijaitsevat kaikki fontit, joita pääasiassa käyttävät kirjastot, jotka suorittavat operaatioita liittyen kuvien manipulointiin.

./system/helpers-hakemistossa on CodeIgniterin mukana tulevat, valmiit auttajatiedostot (helper), joihin on pääsy koko sovelluksella. CodeIgniter ei automaattisesti lataa auttajatiedostoja.

./system/language-hakemistossa on CodeIgniterin ytimen kielitiedostot, joita kirjastot ja auttajatiedostot käyttävät.

./system/libraries-hakemistossa on CodeIgniterin mukana tulevat, valmiit kirjastot. Nämä kirjastot yksilöityvät erilaisiin tehtäviin, kuten esimerkiksi ./system/libraries/Session.php-tiedosto, jonka tarkoitus on helpottaa session käsittelyä sovelluksessa.

./system/logs-hakemistossa on CodeIgniterin generoima lokidata.

./system/plugins-hakemistoon tallennetaan kaikki kolmansien osapuolten tekemät toiminnallisuudet. Tällä tavalla CodeIgniteriin voidaan tuoda esimerkiksi kirjautumistoiminto, joka on jonkun muun kehittäjän ohjelmoima. Pluginit ovat

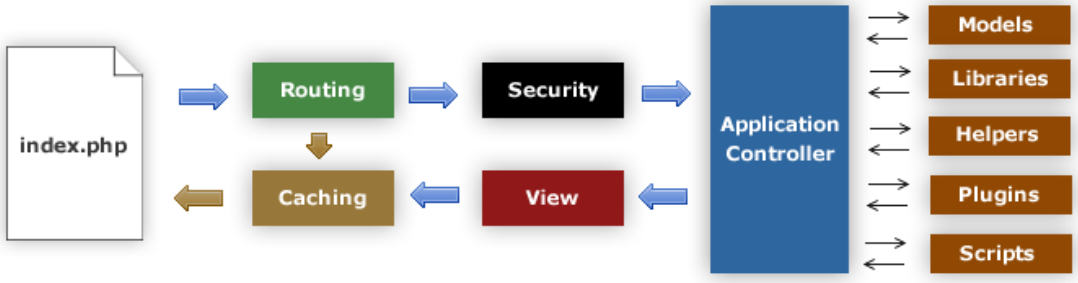
miltei samanlaisia kuin auttajatiedostot, pluginit ova funktioita tai luokkia joita CodeIgniterin yhteisö ylläpitää, kehittää ja jakaa.

./system/scaffolding-hakemisto sisältää luokkia jotka mahdollistavat scaffolding-toiminnallisuuden. Scaffold on metodi joka on kehitetty luomaan kehityksen aikaisia ylläpitoja tietokanta pohjaisiin sovelluksiin. (Tämä ominaisuus on poistunut CodeIgniter versiosta 2.0.0 ylöspäin).

./system/user_guide-hakemistossa on ohjeita CodeIgniterin käyttämiseen.

./index.php-tiedosto sijaitsee sovelluksen juuressa ja kaikki selaimen suorittamat kutsut ohjataan tähän tiedostoon. Index.php tiedoston taas lataa CodeIgniterin ytimen josta sovelluksen suoritus alkaa.

3 MITEN CODEIGNITER TOIMII



Kuvio 2. CodeIgniterin toiminta [viitattu 6.6.2012]. Saatavissa: http://codeigniter.com/user_guide/overview/appflow.html.

CodeIgniterssä kaikki alkaa sovelluksen juuressa sijaitsevasta index.php-tiedostosta. CodeIgniter noudattaa www-kehityksessä yleistä bootsrapping-tekniikkaa. Bootsrapping-tekniikassa sovelluksen yksinkertainen osa aktivoi sovelluksen monimutkaisemmat osat (Wikipedia 2012, Bootsrapping [viitattu 6.6.2012]). CodeIgniterissa sovelluksen yksinkertainen osa on tämä juuri-tason index.php-tiedosto, jonka kautta kaikki selaimen kutsut ohjataan.

Index.php-tiedosto toimii eräänlaisena etukäsittelijänä, joka lataa CodeIgniterin ne ytimen resurssit, jotka vaaditaan sen toimimiseksi. Tämän jälkeen CI_Router-luokka (./system/libraries/Router.php) tutkii ja käsittelee HTTP-pyynnön kertoakseen CodeIgniterille mitä tehdä seuraavaksi. Jos pyydettävä sivu on välimuistissa, lähetetään se selaimelle ja ohitetaan normaali järjestelmän lataus. Jos taas pyydettävä sivu ei ole välimuistissa, CodeIgniter suodattaa HTTP-pyynnön ja kaiken käyttäjän lähettämän tiedon turvallisuustekijöistä ja lataa sovelluksen käsittelijän. Tämän jälkeen käsittelijä lataa mallit, ytimen kirjastot, apurit ja muut resurssit jotka tarvitaan selaimen pyynnön käsittelyä varten. Lopuksi valmis näkymä (view) lähetetään selaimelle näytettäväksi. Jos välimuisti on kytketty päälle, tallennetaan valmis näkymä välimuistiin.

Keskeisenä tekijänä CodeIgniterin logiikassa on kehittäjän luoman sovelluksen käsittelijät ja HTTP-pyyntö, tarkemmin, www-osoite. CodeIgniter poistaa www-

osoitteesta ./application/config/config.php tiedostoon määritetyn sivuston osoitteen. Esimerkki alla kuvaa tätä.

```
<?php
/*
|--------------------------------------------------------------------------
--
| Base Site URL
|--------------------------------------------------------------------------
--
|
| URL to your CodeIgniter root. Typically this will be your base URL,
| WITH a trailing slash:
|
|       http://example.com/
|
| If this is not set then CodeIgniter will guess the protocol, domain and
| path to your installation.
|
*/
$config['base_url']  = 'http://www.sivustonosoite.fi/';
?>

</end of sample code>
```

Jäljelle jäävästä osasta CodeIgniter päättelee mille sovelluksen käsittelijälle HTTP-kutsu ohjataan ja mitä parametreja käsittelijälle annetaan. Esimerkiksi selain kutsuu osoitetta http://www.sivustonosoite.fi/auctions/items. Osoitteesta CodeIgniter poistaa osan www.sivustonosoite.fi. Tämän jälkeen CodeIgniter käyttää jäljelle jääneen osoitteen ensimmäistä osaa (/**auctions**/items) ja etsii hakemistosta application/controllers tiedostoa auctions.php ja sen sisältä luokkaa ”Auctions”. Esimerkki alla kuvaa tätä.

```
<?php if ( ! defined('BASEPATH')) exit('No direct script access al-
lowed');

/**
./application/controllers/auctions.php
*/

class Auctions extends CI_Controller {

 /**
 * Index Page for this controller.
 *
 * Maps to the following URL
```

```
 * http://example.com/index.php/auctions
 *      - or -
 * http://example.com/index.php/auctions/index
 * @see http://codeigniter.com/user_guide/general/urls.html
 */
 public function index()
 {
   $this->load->view('welcome_message');
 }

}
/* End of file auctions.php */
/* Location: ./application/controllers/auctions.php */

</ end of sample code>
```

Kun tiedosto on löytynyt ja sen sisällä on oikea luokka, CodeIgniter etsii www-osoitteesta jäljelle jäänen toisen arvon perusteella (/auctions/**items**) metodia items, jota CodeIgniter kutsuu. Esimerkki alla kuvaa tätä.

```
<?php if ( ! defined('BASEPATH')) exit('No direct script access al-
lowed');

class Auctions extends CI_Controller {
 /**
 * Items Page for this controller.
 *
 * Maps to the following URL
 * http://example.com/index.php/auctions/items
 *      - or -
 * http://example.com/index.php/auctions/items
 * @see http://codeigniter.com/user_guide/general/urls.html
 */
 public function items()
 {
   $this->load->view('items’);
 }

}
/* End of file auctions.php */
/* Location: ./application/controllers/auctions.php */

</ end of sample code>
```

Kyseisessä items-metodissa kehittäjä voi ladata sovelluksen malleja, kirjastoja, auttajia, näkymiä ja hakea tietoa tietokannasta. Tässä esimerkissä kehittäjä

lataa huudettavat tuotteet edustavan mallin (Items_model), kutsuu kyseisen mallin get_where-metodia, sen avulla noutaa kaikki julkaistut huudot tietokannasta ja lataa näkymän välittäen ladattavalle näkymälle tietokannasta haetun tiedon. Esimerkki alla kuvaa tätä.

```
<?php if ( ! defined('BASEPATH')) exit('No direct script access al-
lowed');

class Auctions extends CI_Controller {
/**
 * Items Page for this controller.
 *
 * Maps to the following URL
 * http://example.com/index.php/auctions/items
 *      - or -
 * http://example.com/index.php/auctions/items
 * @see http://codeigniter.com/user_guide/general/urls.html
 */
 public function items()
 {
   $this->load->model('Items');
   $data['items'] = $this->Items->get_where(array(
    'published'=>1,
    'status'=>1)
  );

   $this->load->view('items', $data);
 }
}
/* End of file auctions.php */
/* Location: ./application/controllers/auctions.php */

</ end of sample code>
```

4 HUUTOKAUPPA-ALUSTA

4.1 Projekti

Projekti käynnistyi vuonna 2009. Tilaajana oli Norrolux Oy, joka työskentelee konkurssipesien realisoinnin kanssa ja he tarvitsivat keinon helposti edelleen myydä realisoitua tavaraa. Tätä varten ehdotettiin selaimessa toimivaa huutokauppaa. Ajatus ei kuitenkaan ollut uusi, vaan idea huutokauppaa varten ohjelmoidusta alustasta oli syntynyt jo ennen kuin sen tuottamiselle löytyi tarve asiakkaan toimesta.

Web palvelujen ostajana Norrolux Oy:llä oli vähän kokemusta eikä tästä syystä johtuen projektille asetettu tiukkoja vaatimuksia ominaisuuksien suhteen.

4.2 Alustan kuvaus

Koska huutokaupalle ei tilaajan toimesta asetettu tavoitteita tai määrityksiä, luotiin sille suunnitteluvaiheessa lähtökohdaksi uudelleen käytettävyys. Tavoite oli luoda huutokaupan rakentamisesta helppoa. Oletuksena oli että huutokaupan ydin vaatii paljon monimutkaista logiikkaa, mutta sen päälle pitäisi voida rakentaa koodin tasolla yksinkertainen käyttöliittymä. Kenen tahansa joka osaisi suunnitella www-sivuja tai omaa kokemusta ohjelmoinnista PHP:lla, pitäisi pystyä rakentamaan huutokauppa tällä alustalla.

Huutokauppa-alusta helpottaa huutokaupan rakentamista samalla tavalla kuin ohjelmointikehys helpottaa www-sovellusten rakentamista: se tukee ja nopeuttaa huutokaupan rakentamista tarjoamalla valmiita kirjastoja, toiminnallisuuksia ja funktioita huutokaupassa tarvittaviin yleisiin toimenpiteisiin. Koska huutokauppa-alusta on rakennettu CodeIgniter-ohjelmointikehyksen päälle, tarjoaa se samalla myös www-sovelluksia varten suunnatut työkalut huutokaupan perustajan käyttöön. Huutokauppa-alustaan on valmiiksi integroitu paljon toiminallisuutta, kuten esimerkiksi valmis hallinta, teemojen hallinta, huutojen lisääminen,

huutojen tekeminen ja rekisteröityminen. Huutokauppa-alustan toiminnallisuudet on kokonaisuudessaan kuvattu liitteessä 1.

Käytännössä huutokaupan perustaja määrittää alustaa varten tietokannan, asentaa huutokaupan tietomallin määritettyyn tietokantaan, määrittää huutokaupan www-osoitteen ja huutokauppa on asennettu. Yksinkertaisimmillaan asennus ei vaadi muuta. Silloin jos huutokaupan perustaja haluaa luoda perustettavalle huutokaupalle yksilöllisen rakenteen ja ulkoasun, pitää perustajan osata työskennellä huutokauppa-alustan teemoittamisen kanssa. Käytännössä helpoin tie teemoittamiseen on kopioida alustan mukana tuleva valmis teema (./application/views/content/themes/960/) omaksi hakemistokseen ja aloittaa teeman sisällä olevien PHP-tiedostojen muokkaaminen. Kappaleessa 3.3 Alustan tavoitteet, on tarkemmin kuvattu huutokaupan perustamisen teoriaa.

4.3 Tavoitteet alustalle

Huutokauppa-alustan ensimmäinen tavoite oli käyttöliittymä ja sen muokattavuus. Lähtökohtana ei kuitenkaan ollut se käyttäjän selaimessa näkyvä käyttöliittymä, vaan pinnan alla oleva, ohjelmoijia, suunnittelijoita ja huutokaupan perustajia koskettava käyttöliittymä. Sovelluksen monimutkaisen logiikan päälle piti luoda kerros joka yksinkertaistaa huutokaupalle tyypilliset monimutkaiset toiminnallisuudet ja joka luo hyvin vähän rajoitteita missä mitäkin tietoa tai toiminallisuutta esitetään. Esimerkki alla kuvaa kuinka näkymässä (template), yhdellä funktiokutsulla ladataan yksi huuto. Kaikki tässä kappaleessa näytettävä esimerkki koodi on huutokauppa-alustan auttajatiedostoissa sijaitsevia funktioita, joista suurimalla osalla on pääsy alustan luokkapohjaisten kirjastojen metodeihin.

```
<?php if(item()): $item = get_item(); ?><?php endif; ?>
```

Huutokauppa-alusta tarjoaa rajapinnan huutokaupan kirjastoihin (library) funktionaalisesta ohjelmoinnista rikkoen tällä tavalla hieman MVC arkkitehtuuria.

Huutokauppa-alustan tapauksessa luokattomalla funktiolla on pääsy kaikkiin kirjastossa olevien luokkien julkisiin metodeihin. Tämä päätös oli täysin tietoinen, koska luokkien syntaksi haluttiin pitää poissa näkymiä tuottavista teematiedostoista, tuoden sinne ainoastaan yksinkertaisia funktio kutsuja. Tämä valinta tekee näkymiä luovien teematiedostojen rakentamisesta ja analysoinnista yksinkertaisempaa ja käyttäjäystävällisempää. Tämä päätös myös loi yksinkertaisemman käyttöliittymän sovelluksen ytimen päälle.

Huutokaupan toiminnallisena ytimenä toimivat erilaiset lomakkeet ja niiden luomiseen ja muokkaamiseen haluttiin luoda mahdollisimman yksinkertainen tapa. Jos esimerkiksi kirjautumislomake haluttaisiin etusivulle, pitäisi sen olla mahdollisimman helppoa. Sama koskee lomakkeiden ulkoasua ja rakennetta. Alusta määrittää hyvin tiukasti jokaisen lomakkeen pakolliset kentät, mutta alusta ei saa määrittää lomakkeiden rakennetta visuaalisesti. Huutokaupan perustajan ei myöskään tarvitse huolehtia siitä minne lomake lähetetään, koska alusta huolehtii, että kutsu palautuu aina takaisin paikkaan josta se lähetettiin ja kutsun mukana tieto siitä, onnistuiko operaatio joka lomakkeeseen oli kiinnitetty. Alla oleva esimerkki kuvaa kirjautumislomakkeen lataamista näkymään (template) auttajatiedostossa sijaitsevan login-funktion kautta.

```
<?php _e(login()); ?>
```

Huutokaupassa lomakkeiden lähetykset ja näkymien vaihdot linkittyvät www-osoitteiden verkostoon, jotka taas linkittyvät sovelluksen logiikkaan. Sovelluksen piti myös tarjota yksinkertainen käyttöliittymä tämän verkoston käyttämiseen koodin tasolla. Yksinkertaisuus luo samalla myös mahdollisuuksia, joista yksi on www-osoitteiden muokkaaminen. Osoitteiden muokkaaminen tapahtuisi näkymättömästi taustalla eikä kuitenkaan muuttaisi mitenkään käyttäjälle suunnattua käyttöliittymää. Taustan logiikkaa voidaan muuttaa ilman että se vaikuttaa käyttöliittymän toimintaan. Alla oleva esimerkki kuvaa kuinka sovelluksessa ladataan osoite url-funktion kautta (./application/helpers/as_theme_helper.php), joka osoittaa minne kirjautumislomake lähetetään.

```
<?php _e(url('login_attempt')); ?>
```

Huutokauppaan liittyy paljon tietoa. On käyttäjien tietoja, huutoihin liittyvää tietoa, viestejä, kysymyksiä, kysymyksien vastauksia ja tuotteiden tietoja. Kaiken taustalla on ohjelmallisesti pakotettu relaatiotietokanta, jonka toiminnasta vastaavat sovelluksen kirjastot ja mallit. Tämä tieto ympäröitiin yksinkertaisen käyttöliittymän alle koodin tasolla, jotta huutokaupan perustajalla olisi funktiotason pääsy kaikkeen tietoon joka on valmiiksi jäsennetty, rakenteeltaan loogisiin yksiköihin. Huutokaupan perustaja kutsuu yhtä funktiota ja saa funktiolta esimääritellyn datan, jota käyttää läpi sovelluksen näkymien. Huutokaupan perustajan tarvitsee huolehtia ainoastaan tiedosta jonka hän funktiolta saa ja parametreista joita funktio tarvitsee kutsun yhteydessä. Alla oleva esimerkki kuvaa kuinka yhdelle tuotteelle ladataan siihen liittyvät kysymykset get_questions-funktion kautta.

```
<?php _e(get_questions('markup:TRUE')); ?>
```

Lopullinen tavoite alustaa suunnitellessa oli tehdä huutokaupan perustamisesta yksinkertaista. Asiakkaan pyynnöt rakenteen muutoksesta eivät saisi olla työläitä ja huutokauppaa perustettaessa käyttöliittymä pitäisi pystyä helposti tekemään yksilöllisen näköiseksi.

5 CODEIGNITER JA HUUTOKAUPPA

5.1 Huutokauppa-alustan ydin

Koska CodeIgniter tukee modulaarista suunnittelumallia, huutokauppa-alusta rakennettiin tätä mallia noudattaen. Lähtökohtana oli toteuttaa oma kirjasto ja auttajatiedostot, joiden avulla huutokaupan toiminnallisuus olisi erillään käsittelijöiden logiikasta. Tämä mahdollistaa mm. jokaiselle huutokaupalle yksilölliset www-osoitteet ja oman logiikan. Ylätason tavoite huutokaupan ohjelmoinnissa oli jättää mahdollisimman paljon tilaa suunnanmuutoksille ja ympäröidä kaikki yksilöllinen toiminnallisuus omaan, uudelleen käytettävään kokonaisuuteensa.

Jokainen käsittelijä, malli, näkymä, auttajatiedosto ja kirjasto noudattaa CodeIgniterin logiikkaa ja ohjelmointitapaa. Kaikki auttajatiedostot ovat riippuvaisia CodeIgniterin ytimen metodeista ja funktioista, sama koskee kaikkia malleja ja kirjastoja. Käsittelijöiden lataamisesta huolehtii yksistään CodeIgniter ja ne noudattavat CodeIgniterin määrittämään nimeämistapaa ja ohjelmointitapaa. Käytännössä integraatio CodeIgniteriin on läpinäkyvä ja kaikesta huutokauppa-alustan logiikasta on helppo erottaa mikä on alustan omaa ja mikä CodeIgniterin. Useimmiten tämä näkyy viittauksina $ci-muuttujaan, joka pitää sisällään viittauksen CodeIgniterin ytimen olioon.

Huutokauppa-alustan ytimenä toimii kirjastossa sijaitseva luokka (./application/libraries/As_initializer.php), joka vastaa kaikkien, niin alustan kuin CodeIgniterin resurssien lataamisesta yleisellä tasolla. As_initializer-kirjaston lataamat resurssit määrittää konfigurointi tiedosto, josta löytyvät kaikki kirjastot, mallit ja auttajatiedostot joita tarvitaan huutokaupan toiminnan ylläpitämisessä. Tällä tavalla kaikilla huutokauppa-alustan käsittelijöillä on yksi merkintäpaikka kaikkiin tarvittaviin resursseihin, joka taas yksinkertaistaa käsittelijän logiikkaa.

Esimerkkinä Auctions-käsittelijä joka huolehtii kaikesta huutoihin liittyvästä logiikasta ylätasolla. Auctions-käsitteljän konstruktori lataa kirjastosta ainoastaan

As_initializer-kirjaston, jonka konstruktori huolehtii lopusta. Esimerkki alla kuvaa tätä.

```
<?php


class Auctions extends Controller {

// ------------------------------------------------

/**
 * Auctions
 *
 * Constructor of Auctions controller
 *
 * @access          public
 */
// Constructor

function Auctions()
{

  parent::Controller();

  // Load the As_initializer library
  $this->load->library('as_initializer');
}

/* End of file auctions.php */
/* Location: ./application/controllers/auctions.php */

</ end of sample code>
```

Tällä tavalla uusien käsittelijöiden luonti nopeutuu, koska koko sovelluksella on yksi merkintäpaikka, josta ladata kaikki sovelluksen logiikka. Samalla varmistetaan, että näkymissä olevilla funktio-kutsuilla on kaikki sovelluksen resurssit käytössään siinä vaiheessa kun kontrolli siirtyy näkymille.

Osassa kirjastoja on automatiikkaa joka käynnistyy siinä vaiheessa kun kirjasto ladataan. Esimerkiksi As_user-kirjasto (./application/libraries/As_user.php) omassa konstruktorissaan tarkastaa onko huutoja selaava käyttäjä kirjautunut vai ei. Alla oleva esimerkki kuvaa As_user-kirjaston konstruktoria.

```
<?php
```

```
// ------------------------------------------------------------

/**
* __constructor of As_user-class
*
* _constructor
*
* @access           public
*/
function as_user()
{
  $this->ci =& get_instance();
  $this->ci_killed_user();

  if($this->authenticate())
  {
    $this->_user = $this->ci->user_model->get_user($this->ci->db_session-
>userdata('uid'));
    $this->_log = $this->ci->user_model->get_log($this->_user->id);

    $this->set_data();
    $this->update_log();
 }
 else
 {
   $this->_user = FALSE;
 }
}

 ...

 </ end of sample code>
```

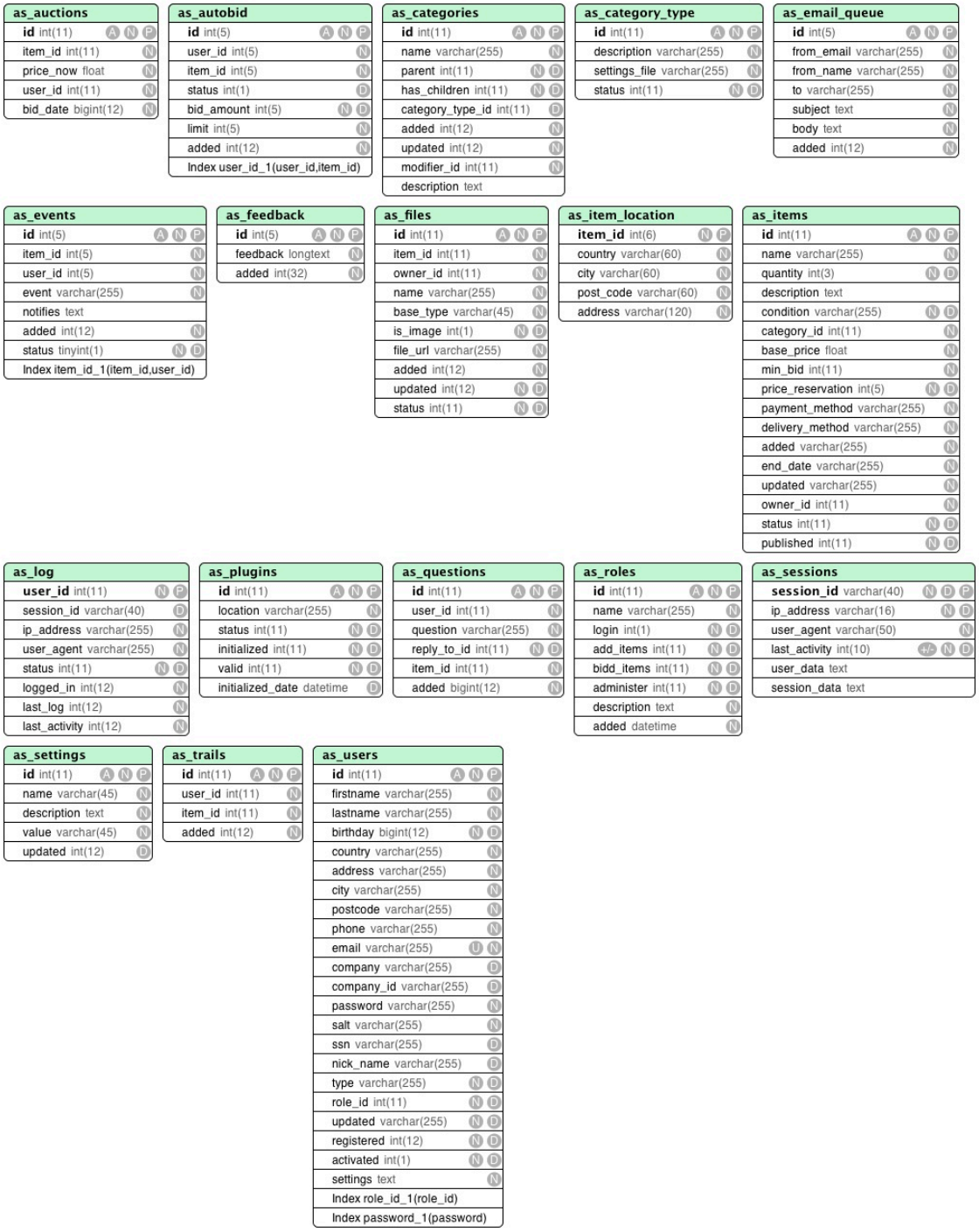
Tällä tavalla sovelluksen käsittelijän käyttöliittymä on entistä yksinkertaisempi. Kun As_intializer-kirjaston konstruktori on ladattu, palautuu kontrolli kutsuvalle käsittelijälle (controller).

Kun huutokauppa-alustan ydin on ladattu, alusta tietää onko käyttäjä kirjautunut vai ei. Alusta tietää myös onko käyttäjä selaamassa listausta huudoista, yhtä huutoa vai esimerkiksi omia tietojaan. Jos käyttäjä on selaamassa yhtä huutoa, on tätä varten automaattisesti haettu kyseisen huudon tiedot tietokannasta odottamaan näkymän pyyntöä näistä tiedoista. Myös kaikki muut alustan kirjastot joilla on oma tietomalli, ovat noutaneet oman tietomallinsa tiedot tietokannasta muita kirjastoja varten. Alustan tekemät oletukset eivät kuitenkaan pakota

huutokaupan perustajaa toimimaan tietyllä tavalla. Alustan tekemillä oletuksilla pyritään saavuttamaan nopeutta ja vähentämään turhia tietokantakyselyitä. Jos huutokaupan perustaja haluaa esittää yhden huudon näkymässä esimerkiksi kaikki aktiiviset huudot, on se mahdollista.

5.2 Alustan tietomalli

Huutokauppa-alustan tietomalli ei ole relaatiotietokanta, vaikka alustassa käytetty MySQL-teknologia mahdollistaakin viiteavaimien käytön. Alustan tietomalli on ohjelmallisesti pakotettu relaatiotietokanta. Ohjelmallisesti pakotettu relaatiotietokanta tarkoittaa, että taulujen välinen yhteys luodaan koodin tasolla siinä vaiheessa, kun alusta luo tietokantakyselyitä tiedon hakemiseen, tallentamiseen, muokkaamiseen ja poistamiseen. Esimerkiksi jos huutokaupasta poistetaan huuto, pitää kyseisen poistettavan huudon ID:llä käydä ns. manuaalisesti poistamassa myös kaikki siihen liittyvä muu tieto, kuten esimerkiksi kysymykset. Koska kysymys on alustasta jossa oletuksena on mm. asennus prosessin yksinkertaisuus, haluttiin tällä tavalla varmistaa yhteensopivuus mahdollisimman monen hosting-ympäristön kanssa.



Kuvio 3. Huutokauppa-alustan tietomalli.

Jotta alustassa voitaisiin käyttää tietokantatason relaatioita MySQL teknologialla, pitäisi tietokantatyypin olla InnoDB. Yleisimmät hosting ympäristöt, kuten perinteiset web-hotellit eivät vakiona tarjoa MySQL:ää InnoDB:n kanssa, koska se kuluttaa enemmän muistia kuin MyISAM, joten palveluntarjoajaa pitäisi aina

huutokauppaa asennettaessa pyytää vaihtamaan InnoDB tietokannan tyypiksi ennen asennusta. Kokemattomalle huutokaupan perustajalle tällainen operaatio oletuksena voi olla liian vaativa, joten huutokauppa-alustassa päädyttiin ohjelmallisesti pakotettuun relaatiotietokantaan.

5.3 Alustan toiminta

5.3.1 Alustan käsittelijöiden toiminta

Kun huutoja selaava käyttäjä kirjoittaa selaimen osoitekenttään www.esimerkkiosoite.com/auctions/single/1, kutsutaan ensimmäiseksi hakemiston juuritasolla olevaa index.php-tiedostoa. Tämä index.php-tiedosto lataa CodeIgniterin ytimen, jossa automaatiot käsittelevät kutsun www-osoitteen. CodeIgniter eristää www-osoitteen osat ja määrittää niistä käsittelijän (www.esimerkkiosoite.com/**auctions**/single/1), metodin (www.esimerkkiosoite.com/auctions/**single**/1) ja mahdolliset parametrit (www.esimerkkiosoite.com/auctions/single/**1**).

Kun CodeIgniter lataa käsittelijän, kutsutaan ensimmäiseksi ladattavan käsittelijän konstruktoria. Alustan käsittelijöiden konstruktorit lataavat kirjastosta (./application/libraries) As_initializer-luokan, jolloin kontrolli siirtyy hetkeksi kirjastoille. Tässä vaiheessa on tarkkaan määritetty mitä ladataan ja missä järjestyksessä. As_initializer-kirjasto lukee application/config-hakemistosta tiedoston as_auctionsite.php, johon on määritetty mallit, auttajatiedostot ja kirjastot jotka pitää ladata ja järjestys jossa ne ladataan.

Ensimmäisenä As_initializer-kirjasto lataa alustan mallit (./application/models), jotta rajapinta tietokantaan on valmiina ennen muita kirjastossa olevien luokkien lataamista. Seuraavana alusta lataa auttajatiedostot. Auttajatiedostot sisältävät ainoastaan luokattomia funktioita joilla on globaali näkyvyys läpi alustan kaikkien osien. Tämän jälkeen As_initializer-kirjasto aloittaa alustan muiden kirjastojen latauksen. Alustan kirjastot on kuvatta liitteessä Huutokauppa-alustan ytimen kirjastot.

Kun kaikki alustan kirjastot on ladattu, kontrolli palautuu alustan ladatulle käsitteljälle, joka palauttaa sen CodeIgniterin ytimelle. Seuraavaksi CodeIgniter etsii ladatusta käsittelijästä www-osoitteesta eristetyllä metodin nimellä vastaavaa metodia. Esimerkiksi yhden huudon www-osoitteella (www.esimerkkiosoite.com/auctions/**single**/1) CodeIgniter varmistaa että Auctions-käsittelijällä on metodi nimeltä single ja kutsuu seuraavaksi sitä. Seuraava esimerkki kuvaa miltä näyttää Auctions-käsittelijän single-metodi.

```
<?php

/**
./application/controllers/auctions.php
*/

// ------------------------------------------------------------

/**
* Single
*
* Loads the template for single item view
*
*/
public function single()
{

  $data['TEMPLATE_PATH'] = 'content/themes/' . $this->_theme . '/';
  $data['view'] = 'single';

  $this->load->view('index', $data);

}
...

?>

</end of sample code>
```

Tässä vaiheessa alustan käsittelijöillä on kahdenlaisia metodeita: näkymiä lataavia metodeita, kuten yllä oleva esimerkki ja logiikkaa suorittavia metodeita, jotka suorittavat jonkin operaation, kuten käyttäjän kirjautumisen. Logiikkaa suorittava metodi sijaitsee aina eri osoitteessa ja uudelleen ohjaa kutsun takaisin osoitteeseen josta kutsu tuli. Esimerkkinä Auctionsite-käsittelijän login_attempt-

metodi, joka suorittaa käyttäjän kirjautumisen ja uudelleen ohjaa käyttäjän takaisin osoitteeseen josta kutsu tuli.

```
<?php

/**
./application/controllers/auctions.php
*/

// ------------------------------------------------------------

/**
* login_attempt
*
* Login user
*
*/
function login_attempt()
{
  $this->as_forms->login(TRUE);

  $this->as_utils->redirect_to($this->as_utils->get_previous_url());
}

...

</ end of sample code>
```

5.3.2 Alustan kirjastojen toiminta

Kirjastot ovat alustan sydän. Niissä suoritetaan huutokaupan monimutkainen logiikka ja useimmat kirjastoista sisältävät automatiikka joka käynnistyy siinä vaiheessa kun kirjasto ladataan. Koska alustan käsittelijät ja funktionaalinen kerros on riippuvainen kirjastojen metodeista ja tiedoista, oletuksena alustan virtauksessa kirjastot ovat ensimmäinen osa automaattista logiikkaa suorittavaa sovellusta joka ladataan.

Kun kirjasto joka sisältää automatiikkaa ladataan, suoritetaan kirjaston konstruktorissa olevat tehtävät ensimmäisenä. Tavallista on että kirjasto pyytää toiselta kirjastolta täydentävää tietoa tai että kirjasto suorittaa hakuoperaatioita tietokantaan jonkin alustan mallin välityksellä. Hyvä esimerkki on As_items-kirjasto (./application/libraries/As_items.php) joka noutaa tietokannasta kaikki

aktiiviset, päättyneet ja julkaisemattomat huudot ja rakentaa saadusta tiedosta muiden kirjastojen avulla valmiin tietokokonaisuuden jokaisesta huudosta ja huutoon liittyvästä tiedosta. Esimerkki alla kuvaa As_items-kirjaston initialize-metodia jota As_items-kirjaston konstruktori kutsuu.

```
<?php

/**
./application/libraries/As_items.php
*/

// ------------------------------------------------------------

/**
* initialize
*
* __constructor
*
* @access           public
*/
function initialize()
{
  $this->_items = new stdClass();

  $data['active'] = $this->ci->item_model->get_active();
  $data['ended'] = $this->ci->item_model->get_where(array('status'=>0));
  $data['unpublished'] = $this->ci->item_model-
>get_where(array('published'=>0, 'status'=>1));

  foreach($data as $key => $value)
  {
    if($value !== FALSE)
    {
      $this->_items->$key = $this->build_data($value);
    }
    else
    {
      $this->_items->$key = null;
    }
  }

  $this->initialize_view();
}
...

</ end of code sample >
```

5.3.3 Alustan mallien toiminta

Alustan mallit, MVC arkkitehtuurin mukaisesti kuuluvat tietomallia käsittelevään kerrokseen. Alustan mallit sisältävät metodeita huutoihin liittyvän tiedon hakemiseen, päivittämiseen, poistamiseen ja lisäämiseen. Teoriassa alustan tietokannan jokaista taulua vastaa oma malli, jonka kautta interaktio tietokannan kanssa tapahtuu. Kirjastot ovat mallien pääasiallisia käyttäjiä ja mallien lataamisesta huolehtii As_initializer-kirjasto (./application/libraries/As_initializer.php).

Alustan mallit ovat niin kutsuttu tietokanta abstraktio, joka tarkoittaa, että interaktio tietokannan kanssa tapahtuu aina samalla syntaksilla ja CodeIgniterin tietokanta-ajurit huolehtivat lopullisesta, matalan tason kyselyiden suorittamisesta teknologiakohtaisesta. Esimerkki huutojen mallin (./application/models/item_model.php) update_where-metodista, jossa päivitetään yhden huudon tiedot ja joka osoittaa millainen on CodeIgniterin tietokanta abstraktion syntaksi.

```
<?php

/**
./application/models/item_model.php
*/

// ------------------------------------------------------------

/**
* update_where
*
* Updates a items data
*
* @access public
*/

function update_where($where, $data)
{
  $this->db->where($where);
  $this->db->update('as_items', $data);

}

...

</ end of sample code >
```

5.3.4 Alustan funktionaalinen kerros

Alustalla on teemoihin, käsittelijöihin ja kirjastoihin linkittyvä funktionaalinen kerros, joka muodostuu auttajatiedostoista. Vaikka alustan funktionaalisen kerroksen vaikutus heijastuu läpi koko alustan, on sen merkittävin käyttäjä kuitenkin alustan kerros joka työskentelee näkymien kanssa, eli näkymät (view).

Funktionaalinen kerros ei sisällä automaatiota, eikä ilman käskyä suorita mitään ja on tällä tavalla puhtaasti tarkoitettu auttamaan pienten operaatioiden tekemisessä. Tärkein tehtävä tällä funktionaalisella kerroksella on toimia rajapintana alustan näkymien ja kirjastojen välillä. Auttaja tiedostoissa sijaitsevilla funktioilla on pääsy kirjastossa sijaitsevien luokkien julkisiin metodeihin, tuoden tällä tavalla paljon työkaluja teematiedostojen tasalle. Teematiedostossa saattaa näkyä esimerkiksi alla olevan esimerkin kaltainen kohta.

```
<?php foreach(get_items() as $item):  set_item($item->id); ?>
```

Yllä oleva esimerkki osoittaa miten teematiedostosta on pääsy funktionaalisen kerroksen kautta As_items-kirjastoon, jolta get_items-funktio pyytää kaikkia avoimia huutoja. Tässä esimerkissä alusta osoittaa miten yksinkertainen on koodin tasolla oleva käyttöliittymä, jota huutokaupan perustaja käsittelee. Alustan As_initializer-kirjasto on huolehtinut että kaikki resurssit on ladattu ennen kuin kontrolli siirtyy näkymien kerrokseen, joten oletuksena funktionaalinen kerros on ladattu ja käytössä läpi alustan kaikkien kerrosten. Alla esimerkki miltä näyttää ./application/helpers/as_items_helper.php-tiedostossa sijaitseva get_items-funktio.

```
<?php

/**
./application/helpers/as_items_helper.php
*/

// ------------------------------------------------------------

/**
```

```
* get_items
*
*
* @access           public
*/
if ( ! function_exists('get_items'))
{
  function get_items($property = null)
  {
    global $ci;

      if(!is_object($ci))
      {
        $ci = load();
      }

      $items = $ci->as_items->get();

      if(isset($items))
      {
         if(isset($property))
         {
           if(isset($items->$property) && $items->$property !== FALSE)
           {
             return $items->$property;
           }
           else
           {
              die('System error.');
           }
      }
      else
      {
        if(isset($items->active))
        {
          return $items->active;
        }
        else
        {
          die('System error.');
        }
     }
   }
   else
   {
     return null;
   }
  }
}
...

</ end of sample code>
```

5.3.5 Alustan teemojen toiminta

Huutokaupan visuaalinen käyttöliittymä rakentuu näkymistä, jotka on pilkottu teematiedostoiksi. Jokaisella näkymällä voi olla esimerkiksi header-, sivupalkki-, sisältö-, ja footer-teematiedosto. Teematiedostot ovat PHP-tiedostoja, jotka sisältävät PHP:ta ja HTML:ää. Alustan teemat taas ovat yksilöllisesti nimettyjä hakemistoja ./application/views/content/themes-hakemistossa ja käytettävä teema on tallennettu alustan asetuksiin (tietokantaan).

Alustan näkymien lataamisesta ylätasolla huolehtii ./application/views-hakemiston juuressa sijaitseva index.php-tiedosto. Alustan käsittelijät (controller) jotka tarvitsevat näkymiä, lataavat CodeIgniterin view-metodin kautta ./application/views-hakemiston juuressa sijaitsevan index.php-tiedoston ja välittävät sille tiedon siitä mikä teema on alustalla käytössä ja mikä näkymä pitää ladata. Kyseinen tieto tallennetaan aina käsittelijän toimesta $view- ja $TEMPLATE_PATH muuttujiin. ./application/views-hakemiston juuressa sijaitseva index.php tiedosto lataa annetun tiedon perusteella oikeasta teemasta oikean näkymän. Alla oleva esimerkki kuvaa ./application/views/index.php-tiedostoa.

```
<?php

/**
./application/views/index.php
*/

$this->load->view($TEMPLATE_PATH . $view);

?>

</ end of sample code>
```

CodeIgniterin view-metodi huolehtii teematiedostojen lataamisesta ja tarvitsee tätä varten kaksi parametria. Ensimmäinen parametri on teematiedoston nimi, jota view-metodi etsii, jos ei erikseen ole määritetty,niin hakemistosta ./application/views. Toinen parametri on tieto joka välitetään ladattavalle teematiedostolle. Välitettävän tiedon pitää olla PHP taulukko muuttuja, jonka CodeIg-

niter käsittelee. Alustan käsittelijöiden metodit jotka lataavat näkymiä alustavat aina taulukko muuttujan, johon on määritetty kaksi solua: ‘view’- ja ‘TEMPLATE_PATH’. Alla esimerkki miten alustan käsittelijät alustavat kyseiset muuttujat.

```
<?php

$data[
  ‘view’ => ‘single’,
  ‘TEMPLATE_PATH’ => content/themes/' . $this->_theme . '/'
];

?>

</end of sample code>
```

CodeIgniterin view-metodi muuttaa sille parametrina annetun taulukko muuttujan yksittäisiksi, taulukon soluja vastaaviksi arvoiksi. Alla olevassa esimerkissä kuvataan miten yllä olevan esimerkin taulukko muuntuu CodeIgniterin view-metodissa ennen kuin se luovutetaan itse teema tiedostolle.

```
$data[‘view’] → $view
$data[‘TEMPLATE_PATH’] → $TEMPLATE_PATH

</ end of sample code>
```

Kun välitettävä tieto saapuu teematiedostolle, ei se enää ole taulukko, vaan yllä olevan esimerkin mukaisesti voidaan käsitellä yksittäisiä muuttujia. Tämä toiminnallisuus on tärkeää ymmärtää, koska se on olennainen osa CodeIgniterin toimintaa.

Esimerkkinä yhden huudon näkymän lataaminen. Yhden huudon näkymää muodostettaessa, Auctions-käsittelijän konstruktori pyytää As_settings-kirjastolta (./application/libraries/As_settings.php) tiedon käytettävästä teemasta ja tallentaa sen omaan yksityiseen $_theme-muuttujaan, jotta alustuksen jälkeen Auctions-käsittelijän kaikilla metodeilla olisi pääsy tähän tietoon. Kyseisen käsittelijän single-metodi lukee teeman tiedon $data[‘TEMPLATE_PATH’] muuttujaan ja tallentaa ladattavan teematiedoston nimen $data[‘view’]-muuttujaan.

Tämä tieto välittyy ./application/views-hakemiston juuressa sijaitsevalle index.php-tiedostolle CodeIgniterin view-metodin kautta. Käytännössä ./application/views-hakemiston juuressa sijaitseva index.php toimii siltanan alustan käsittelijöiden ja teemojen välillä, ainoastaan edelleen ladaten sen mitä käsittelijät siltä pyytävä.

Alustan teema-kerros on käyttöliittymältään eniten muokattavissa. Käytännössä huutokaupan perustaja voi itse päättää käyttämänsä teeman sisällön, mutta alustaan on myös integroitu niin kutsuttu default-teema, jonka nimi on 960. Tämä teema pitää sisällään kaikki tarvittavat teematiedostot ja samalla antaa mallin, millainen alustan teeman rakenne voi olla. Yksilöllinen teema sisältää neljänlaisia tiedostoja: layout-tiedostoja, partiaali-tiedostoja, sähköposti-tiedostoja ja snippet-tiedostoja.

Layout-tiedosto määrittää näkymän rakenteen, toimien ikään kuin palapelin pohjana, joka kootaan pienistä staattisista ja dynaamisista palasista. Layout-tiedoston rakenne riippuu ladattavasta näkymästä ja jokaisen layout-tiedoston sisältö on täysin huutokaupan perustajan määriteltävissä. Alla esimerkki “Unohditko salasanasi” näkymää palvelevasta ./application/views/content/themes/960/forgot-password.php-tiedostosta, joka kuuluu layout-tiedostojen joukkoon.

```
<?php
/**
./application/views/content/themes/960/forgot-password.php
*/

?>
<?php $this->load->view($TEMPLATE_PATH . 'header'); ?>

<div class="container_16">

  <div id="content">

    <div class="grid_3">
      <?php $this->load->view($TEMPLATE_PATH . 'sidebar-site'); ?>
    </div>

    <div class="grid_9">

      <div id="forgot-password-container">
```

```
          <div class="forgot-password-canvas">

            <h2>Unohditko salasanasi?</h2>

            <?php _e(status_message()); ?>

            <p>Anna sähköpostiosoitteesi niin lähetämme sinulle uuden.</p>

            <?php _e(retrieve_password_form()); ?>

          </div>

    </div>

</div><!-- END OF .grid_9 -->

</div><!-- END OF #content -->
</div>

<?php $this->load->view($TEMPLATE_PATH . 'footer'); ?>

</ end of sample code>
```

Yllä olevassa, yksinkertaisessa esimerkissä näkyy kuinka palapeli muodostuu. ./application/views/content/themes/960/forgot-password.php tiedosto lataa kolme eri partiaali-tiedostoa CodeIgniterin view-metodin avulla: head-partiaalin, sidebar-site-partiaalin ja footer-partiaalin. Sen lisäksi ./application/views/content/themes/960/forgot-password.php käyttää alustan funktio-kirjastoja, jotka sijaitsevat auttajatiedostoissa, ladatakseen mahdolliset viestit järjestelmästä ja lomakkeen jolla käyttäjä voi pyytää uutta salasanaa.

Partiaali-tiedosto on näkymiä muodostavassa palapelissä enemmän staattista sisältöä esittävää kerrosta, kuten esimerkiksi sivun niin kutsuttu footer, sivupalkki- tai head-osio ja siksi usein koostuu puhtaasta HTML:stä. Partiaali-tiedostoilla huutokaupan perustaja voi esimerkiksi sivupohja kohtaisesti ladata eri head-osion, joka taas mahdollistaa esimerkiksi osiokohtaisen ulkoasun ja metatiedon. Alustan mukana tulevassa teemassa on käytössä yksi head-partiaali, josta osa esimerkissä alla.

```
<!--./application/views/content/themes/960/header.php -->

<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Huutokauppa</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="copyright" content="" />
<meta name="description" content="" />
<meta name="keywords" content="" />
<meta name="author" content="" />
<meta name="robots" content="index,follow" />
<meta name="" content="" />
<link rel="shortcut icon" href="" type="image/x-icon" />

<?php _e(meta_jquery()); ?>
<?php _e(meta_jquery_ui()); ?>
<script type="text/javascript" src="<?php _e(base_url() . 'as-
sets/js/jquery-1.3.2.min.js'); ?>"></script>
<link rel="stylesheet" media="screen" type="text/css" href="<?php
_e(theme_url() . '/style.css'); ?>">
<link href="<?php _e(base_url() . 'as-
sets/js/colorbox/style1/colorbox.css'); ?>" media="screen"
rel="stylesheet" type="text/css"/>

<script type="text/javascript" src="<?php _e(base_url() . 'as-
sets/js/colorbox/jquery.colorbox.js'); ?>"></script>
<script type="text/javascript" src="<?php _e(theme_url() .
'/js/theme.js'); ?>"></script>

<!--[if IE]><link rel="stylesheet" type="text/css" href="<?php
_e(theme_url() . '/ie.css'); ?>"><![endif]-->
<!--[if IE 7]><link rel="stylesheet" type="text/css" href="<?php
_e(theme_url() . '/ie7.css'); ?>"><![endif]-->
<!--[if IE 6]><link rel="stylesheet" type="text/css" href="<?php
_e(theme_url() . '/ie6.css'); ?>"><![endif]-->

</head>

...

</ end of sample code>
```

Snippet-tiedostot ovat iso osa alustan dynaamista sisältöä. Mm. alustan kaikki lomakkeet sijaitsevat näissä snippet-tiedostoissa. Snippet-tiedostot sisältävät PHP:ta ja HTML:ää. Snippet-tiedostot ovat alustalle elintärkeitä, koska ne on integroitu syvälle alustan funktionaaliseen kerrokseen ja kirjastojen logiikkaan. Tässä mielessä ne poikkeavat hieman alustan implementoimasta ajattelutavasta, pakottaen jokaisen uuden teeman pitämään sisällään snippets-hakemiston

ja kaiken sen sisällön. Alusta ei määritä mitä snippet-tiedostot pitävät sisällään, joten niitä voi muokata samalla tavalla kuin muitakin näkymä kerroksen tiedostoja. Snippet tiedostot ladataan samalla tavalla kuin alustan muut näkymäkerrokseen kuuluvat tiedostot, CodeIgniterin view-metodin avulla ja usein myös ottavat vastaan parametrina tietoa jonka snippet esittää. Hyvä esimerkki snippet-tiedostojen käytöstä on alustan mukana tulevan teeman single.php-tiedosto, joka esittää yhden huudon näkymää. Yhden huudon näkymässä mm. kysymyslomake, tehtyjen huutojen listaus, huudon asettamisen lomake ja kuvat tuotetaan snippet-tiedostoista. Alla olevassa esimerkissä kuvataan snippet-tiedostoa joka generoi kysymyslomakkeen, tuotteelle asetetut kysymys/vastaus parit ja lomakkeen jolla voi vastata kysymykseen ja kohdan ./application/views/content/themes/960/single.php-tiedostosta joka lataa kyseisen snippet-tiedoston.

```
<?php
/**
./application/views/content/themes/960/single.php - line 58 – 62
*/
?>

<?php if(item_has('questions')): ?>

  <?php _e(get_questions('markup:TRUE')); ?>

<?php else: ?>

  <p>Ei esitettyj&auml; kysymyksi&auml;.</p>

<?php endif; ?>

</ end of sample code>


// /application/views/content/themes/960/snippets/question.php
<div id="question-<?php _e($question->id); ?>" class="question">

  <p class="who"><small><?php _e($question->nick_name); ?> kysyi <?php
_e(format_time($question->added)); ?></small></p>

  <p class="what"><?php _e($question->question); ?></p>

  <?php if(question_has_replies($question->id)): ?>
```

```
    <?php foreach(get_replies($question->id) as $reply): ?>

      <div class="reply">

        <p class="who"><small><?php _e($reply->nick_name); ?> vastasi
tähän kysymykseen <?php _e(format_time($reply->added)); ?></small></p>

        <p class="what"><?php _e($reply->question); ?></p>

      </div>

    <?php endforeach; ?>

  <?php endif; ?>

<?php if(!question_has_replies($question->id)): ?>

  <?php if(isset($user) && $user->id == $item->owner_id): ?>

    <p><a href="javascript:;" class="reply-to-question" id="question-
<?php _e($question->id); ?>">Vastaa tähän kysymykseen</a></p>

    <div id="reply-to-<?php _e($question->id); ?>" class="reply-to"
style="display:none;">

      <?php _e(reply_to($question->id)); ?>

    </div>

  <?php endif; ?>

<?php endif; ?>

</div>

</ end of sample code>
```

Sähköpostitiedostot tai ymmärrettävämmin sähköpostipohjat, snippet-tiedostojen tapaan poikkeavat alustan ajattelutavasta olemalla pakollinen hakemisto jokaisessa teemassa. Sähköpostitiedostoja käytetään kun alusta lähettää viestejä rekisteröityneille käyttäjille, kuten esimerkiksi ilmoitus rekisteröitymisen vahvistamisesta. Tässäkin tapauksessa alusta ottaa ainoastaan kantaa hakemiston ja hakemistossa olevien tiedostojen olemassaoloon, ei niiden sisältöön, joka on täysin huutokaupan perusajan muokattavissa. Sähköposti-

tiedostot sisältävät HTML:ää ja PHP:ta. Alla esimerkki sähköpostitiedostosta jonka alusta lähettää kun käyttäjä rekisteröityy.

```
<?php
/**
./application/views/content/themes/960/emails/confirmation.php
*/
*>


<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title><?php echo $site_name; ?> - Vahvistathan rekisteröitymis-
esi.</title>
<style type="text/css">

body {
background-color: #ffffff;
font-size: 12px;
line-height: 1.5em;
font-family: Lucida Sans, Arial, sans-serif;
color: #312f2f;
}

h2 {
font-size: 16px;
color: #312f2f;
}

a {
text-decoration: none;
color: #017290;
}

p {
font-size: 12px;
}

</style>
</head>
<body style="background-color: #ffffff; font-size: 12px; line-height:
1.5em; font-family: Lucida Sans, Arial, sans-serif; color: #312f2f; mar-
gin: 0; padding: 0;" >

<table width="100%" cellpadding="20" cellspacing="0"
class="backgroundTable" bgcolor="#ffffff" >
<tr>
<td align="center">

<table width="400" cellpadding="10" cellspacing="0"
class="backgroundTable" bgcolor="#ffffff" >
<tr>
```

```
                                                <td><h2>Terve!
Rekister&ouml;idyit <?php echo $site_name; ?> huutokauppa sivustolle,
t&auml;m&auml; on sinun tilis aktivointi viestisi.</h2>
</tr>
<tr>
<td>
                                                        <p>Alla on ak-
tivointi linkki jotka klikkaamalla aktivoit tilis ja voit kirjautua
sis&auml;&auml;n huutokauppan.</p>
</td>
</tr>
<tr>
<td>
                                                        <p>Huomioithan
ett&auml; huutokaupan yll&auml;pit&auml;j&auml; on
m&auml;&auml;r&auml;nnyt sinulle roolin joka saatta rajoittaa huutokauppa

        k&auml;ytt&auml;ytymist&auml;. Alla kuvaus roolista joka on
sinulle asetettu toistaiseksi:</p>
</td>
</tr>
<tr>
<td>
                                                        <p><?php echo
$role_description; ?></p>
</td>
</tr>
<tr>
<td><a href="<?php echo $activation_url; ?>" target="_blank" ti-
tle="Aktivoi tilis">
                                                        Klikkaamalla
t&auml;st&auml; p&auml;&auml;set aktivoimaan tilisi.</a></td>
</tr>
<tr>
<td>

        <p>Lis&auml;tietoja s&auml;hk&ouml;postiosoitteesta: <?php echo
$reply_to; ?></p>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td align="center">
<table width="400" cellpadding="10" cellspacing="0"
class="backgroundTable" bgcolor="#ffffff" >
<tr>
<td><a href="<?php echo $site_url; ?>" title="<?php echo $site_name; ?>"
target="_blank"><?php echo $site_name; ?></a></td>
</tr>
</table>
        </td>
</tr>
</table>
</body>
</html>
```

```
</ end of sample code>
```

Se miten näkymä kerros on teoreettisesti jaettu neljään eri kategoriaan ei käytännössä vaikuta huutokaupan perustamiseen. Teoreettinen kategorisointi helpottaa alustan käyttäjälle näkyvän kerroksen toiminnan dokumentointia ja avaa myös web-kehityksessä hyvin yleistä ilmiötä, jossa näkymä rakennetaan palapelin tavoin useista erilaisista tiedostoista. Näkymä kerroksen jakaminen palapelin tavoin useaan eri osaan antaa huutokaupan perustajalle myös vapauden valita ja tehdä suunnan muutoksia. Tämä myös tukee alustan implementoimaa ajattelutapaa jossa tehdään mahdollisimman vähän oletuksia.

6 PROJEKTIN ANALYSOINTI

Sovelluksen lopullinen formaatti on hyvin kaukana siitä mistä sen ohjelmointi alkoi. Niin rakenteeltaan kuin ajattelutavaltaan lopullinen versio koki koko koodipohjan uudelleen kirjoituksen ennen lopullista muotoaan. Alustan ohjelmointi alkoi ilman opinnäytetyössä dokumentoitua CodeIgniter-ohjelmointikehystä. Alkuperäinen tavoite oli ohjelmoida oma kehys ja sen päälle huutokauppa-alusta. Ajan kuluessa tavoitteen työmäärän todellisuus alkoi paljastua ja samaan aikaan tutustuminen ohjelmointikehysten teknologiaan ja ajattelutapaan saavutti pisteen, jossa niiden käyttöönotto tuntui varteen otettavalta vaihtoehdolta. Tässä vaiheessa oman kehyksen ominaisuudet tukivat jo tietokanta abstraktiota, objektien lataamista singleton-metodin avulla ja session hallintaa. Kuitenkin oman kehyksen koodipohja alkoi hiljalleen paljolti muistuttaa jo olemassa olevien kehysten implementoimaa mallia, joten siirtyminen valmiin kehyksen käyttöön tuntui järkevältä.

Lopulliseksi kehykseksi valittiin CodeIgniter ja nyt ajatellen, valinta oli mielestäni oikea. Valmis ohjelmointikehys on hyvin dokumentoitu ja testattu. Sillä on yhteisön tuki joka myös osaltaan varmistaa kehyksen jatkuvan kehityksen. CodeIgniterin oli tuottanut EllisLab niminen yritys jolla on myös vahva kaupallinen tausta, joka taas osaltaan luo ammattimaisen lähestymisen moneen web kehityksessä vastaan tulevaan ongelmaan. Siirtymä omasta kehyksestä CodeIgniteriin oli melko helppo ja suoritettu muutamassa päivässä. Tämän jälkeen alustan koodipohja koki vielä yhden ison muutoksen.

CodeIgniteriin implementoidussa ensimmäisessä alustan versiossa kaikki logiikka oli ohjelmoitu suoraan käsittelijöihin. Todellisuus kuitenkin oli, että kyseinen lähestymistapa sulki liikaa portteja suunnanmuutoksilta, joten kaikki logiikka päätettiin siirtää kirjastoihin. Tätä tukemassa oli CodeIgniterin implementoima mahdollisuus luoda kirjastoja ja ladata niitä käsittelijöistä käsin. Tämän jälkeen alustan koodipohja ei enää kokenut isoja muutoksia.

Projektin lopputulos ei kaikilta osin vastannut kehittäjänsä odotuksia. Lähtökohdiltaankin jo iso projekti kärsi osittain ajanpuutteesta. Kokonaisuudessaan varovaisesti arvioitu aika jona sovellusta rakennettiin oli noin 1,5 vuotta. Sovelluksen kehityskaari seurasi hyvin tiiviisti oman kehittäjänsä kehityskaarta ja tästä johtuen niin kutsuttu koodipohja koki isoja muutoksia versioiden välillä useampaan otteeseen. Opinnäytetyössä dokumentoitu alustan versio jäi myös ominaisuuksiltaan vajavaiseksi. Iso osa määritellyistä huutokauppa-alustan ominaisuuksista oli ajanpuutteesta johtuen pakko jättää pois, kuten esimerkiksi automatisoitu asennus ja haku. Alusta kärsii myös yhdestä isosta puutteesta, joka on elintärkeä kaikille isoille sovelluksille: dokumentointi. Arviolta 80% huutokauppa-alustan koodista on kommentoimatta ja tällä tavalla puutteellinen. Tärkein lopputulos koko projektissa oli kuitenkin kokemus ja tietotaito joka on vienyt kehittäjäänsä eteenpäin myös työelämässä. Ajatuskin on hieno ja huutokauppa-alusta tulee vielä näkemään lisää versioita.

LÄHTEET

Wikipedia 2012, MVC-arkkitehtuuri [viitattu 6.6.2012]. Saatavissa: http://fi.wikipedia.org/wiki/MVC-arkkitehtuuri

Wikipedia 2012, Bootsrapping [viitattu 6.6.2012]. Saatavissa: http://en.wikipedia.org/wiki/Bootstrapping#Computing

Huutokauppa-alustan ytimen kirjastot

Huutokauppa-alustan ytimen kirjastot ovat php-luokkia, jotka sijaitsevat hakemistossa ./application/libraries ja noudattavat CodeIgniterin määrittämää nimeämismallia.

1. As_initializer-kirjasto

Tämän kirjaston pääasiallinen tehtävä on ladata kaikki sovelluksen resurssit sen perusteella mitä on määritetty hakemistossa ./application/config/ olevaan auctionsite.php tiedostoon. Kyseiseen tiedostoon on As_initializer-kirjastoa varten määritetty mitä kirjastoja, malleja ja apureita pitää ladata. Alla oleva esimerkki kuvaa auctionsite.php-tiedoston kirjastojen, mallien ja auttajatiedostojen määrityksiä:

```
<?php

/**
./application/config/auctionsite.php
*/

/*
|
|       Define helpers to load on initialization
|
*/
$config['helpers'] = array('as_theme', 'as_auction', 'as_item',
'as_category', 'as_file', 'as_user', 'as_question', 'as_forms',
'as_utils', 'as_events');

/*
|
|       Define admin helpers to load on initialization
|
*/
$config['admin_helpers'] = array('admin', 'email', 'admin_forms',
'admin_auction', 'admin_item', 'admin_category', 'admin_file',
'admin_user', 'admin_question', 'as_utils', 'admin_events');

/*
|
|       Define libraries to load on initialization
|
*/
```

```
$config['libraries'] = array('as_password', 'as_settings', 'as_forms',
'as_utils', 'as_auctionsite', 'as_files', 'as_questions',
'as_categories', 'as_auctions', 'as_users', 'as_items', 'as_user',
'as_emails', 'as_events', 'as_validator', 'as_autobid',
'as_maintenance');

/*
|       Define modesl to load on initialization
|
*/
$config['models'] = array('auction_model', 'user_model', 'item_model',
'category_model', 'file_model', 'settings_model', 'question_model',
'autobid_model');

</ end of sample code>
```

Jokainen kirjasto, malli ja auttajatiedosto ladataan CodeIgniterin load-luokan metodien avulla. Alla oleva esimerkki kuvaa tätä:

```
<?php
/**
./application/libraries/As_initializer.php
*/

// ------------------------------------------------------------

/**
* load_libraries
*
* Load all defined libraries
*
*/
  function load_libraries()
  {
                    $libraries = $this->CI->config->item('libraries',
'auctionsite');

                    foreach($libraries as $library)
                    {
                              $this->CI->load->library($library);
                    }
  }

...

</ end of sample code>
```

Jokainen käsittelijä (controller) yhdessä käsittelijään kuuluvien metodien kanssa linkittyvät selaimen lähettämän kutsun www-osoitteeseen.

2. As_password-kirjasto

As_password-kirjasto tarjoaa As_user-kirjastolle toiminallisuuden salasanojen käsittelyyn. Tämä kirjaston tehtävä on generoida turvallisia salasanoja rekisteröitymistä varten ja muodostaa tietokannassa olevia salasanoja vastaavia salasanoja annettujen parametrien perusteella. Alla oleva esimerkki kuvaa jälkimmäistä tapausta:

```
<?php

/**
./application/libraries/As_password.php
*/

// ------------------------------------------------------------

/**
* get_password
*
* Generates matching password for login action
*
* @access           public
* @param            obj         user object
* @param            string      password from login form
* @return           string hashed password for database query
*/
function get_password($user, $password)
{
        $combine = $user->email . $password . $user->salt;

        return md5($combine);
}
...

</ end of sample code>
```

3. As_settings-kirjasto

As_settings-kirjasto luo rajapinnan sovelluksen tietokantaan tallennettuihin asetuksiin. Toiminnallisuuksiltaan hyvin yksinkertainen kirjasto, mutta kriittinen sovelluksen muille kirjastoille. Pitää sisällään metodit asetuksien hakemiseen, tallentamiseen ja päivittämiseen. Esimerkiksi jos jokin muu kirjasto haluaa tietää mitä teemaa sovellus käyttää, voi sitä pyytää As_settings-kirjastolta. Alla ole-

vassa esimerkissä Auctions-käsittelijä (./application/controllers/auctions.php) omassa konstruktorissaan pyytää As_settings-kirjastolta theme-nimistä asetusta:

```
<?php

/**
./application/libraries/As_settings.php
*/

// ------------------------------------------------------------

/**
* __constructor
*
* @access           public
*/
function Auctions()
{

        parent::Controller();

        $this->load->library('as_initializer');

        // Load the theme settings from database
        $this->_theme = $this->as_settings->get('theme');
}

...

</ end of sample code>
```

4. As_forms-kirjasto

As_forms-kirjasto on yksi sovelluksen tärkeimmistä kirjastoista. Ylätasolla As_forms-kirjasto huolehtii kaikkien lomakkeiden generoinnista, validoinnista, uudelleen ohjauksesta ja tehtävien kartoittamisesta eteenpäin.

As_forms-kirjasto rakentaa ja validoi lomakkeita sen perustella mitä on määritetty ./application/config/theme.php-tiedostoon. Kyseiseen tiedostoon on kuvattu lomakkeen kaikki tarvittavat attribuutit ja säännöt, kuten myös virhe viestien sisältö. Alla oleva esimerkki kuvaa kirjautumislomakkeen määrityksiä ./application/config/theme.php-tiedostossa:

```
<?php
```

```
/**
./application/config/theme.php
*/

$config['login_form'] = array(
        'email' => array(
        'attr'=>'name:email|id:email|size:12|value:null|data:null',
        'vldt'=>'required:TRUE|type:text|format:str|min:3|max:255',
        'emsg'=>'empty:Sähköposti kenttä on tyhjä.|min:Jätit tämän kentän
tyhjäksi.|max:null|type:null'
        ),
        'password' => array(
        'attr'=>'name:password|id:password|size:12|value:null|data:null',
        'vldt'=>'required:TRUE|type:text|format:str|min:3|max:255',
        'emsg'=>'empty:Salasana kenttä on tyhjä.|min:Jätit tämän kentän
tyhjäksi.|max:null|type:null'
        ),
);

...

</ end of sample code>
```

5. As_utils-kirjasto

As_utils-kirjastoa tarvitaan läpi koko sovelluksen vaikka onkin toiminnallisuuksiltaan suppea. Pääsiassa tämä kirjasto työskentelee sovelluksen sisäisten www-osoitteiden kanssa. As_utils-kirjasto myös tallentaa jokaisen kutsun tapahtuessa edellisen www-osoitteen josta kutsu tuli. Kyseisestä toiminnosta mm. sovelluksen käsittelijät (controller) ovat riippuvaisia. Alla oleva esimerkki kuvaa User-käsittelijän update_settings-metodia, joka tallentaa käyttäjän asettamat asetukset ja kutsuu As_utils-kirjaston metodia get_previous_url uudelleen ohjatakseen käyttäjän sivulle josta kutsu tuli.

```
<?php

/**
./application/libraries/As_utils.php
*/

function update_settings()
{
        $this->as_forms->edit_user_settings(TRUE);

        redirect($this->as_utils->get_previous_url());
}
```

```
...

</ end of sample code>
```

6. As_auctionsite-kirjasto

As_auctionsite-kirjastolla on ainoastaan kaksi metodia: get_message ja set_message. As_auctionsite-kirjasto tallentaa ja palauttaa viestejä, joita muut sovelluksen osat pyytävät. Esimerkkinä käyttäjän kirjautuminen. Jos kirjautuminen onnistuu asetetaan käyttäjää varten viesti, joka näytetään käyttäjälle seuraavan näkymän latautumisen yhteydessä ja tässä viestissä kerrotaan että kirjautuminen onnistui. Alla oleva esimerkki kuvaa As_user-kirjaston login-metodia, jossa onnistuneesta kirjautumisesta asetetaan viesti käyttäjälle As_auctionsite-kirjaston kautta.

```
<?php

/**
./application/libraries/As_auctionsite.php
*/

// As_user-kirjaston login metodin kohta jossa viesti asetetaan

$this->ci->as_auctionsite->set_message('login', TRUE, 'Olet kirjautunut
sisään.');

...

</ end of sample code>
```

7. As_files-kirjasto

As_files-kirjasto huolehtii sovelluksen huutoihin liittyvistä kuvista. Uutta huutoa lisättäessä kuvien lataamisesta ja käsittelystä huolehtii tämä kirjasto. As_files-kirjasto myös huolehtii kuviin liittyvän tiedon noutamisesta tietokannasta ja tämän tiedon palautuksesta muulle sovellukselle. Alla oleva esimerkki kuvaa As_files-kirjaston metodia get_files, joka palauttaa perustuen saamaansa parametriin, joko kaikki kuvat tai tiettyyn huutoon liittyvät kuvat.

```
<?php

/**
./application/libraries/As_files.php
*/

function get_files($id = null)
{
        if(isset($this->_files)
        && count($this->_files) > 0
        && is_array($this->_files))
        {
                    $files = $this->_files;

                    if(isset($id))
                    {
                                $tmp_arr = new stdClass();
                                $count = 0;
                                $found = FALSE;

                                foreach($files as $file)
                                {
                                            if($file->item_id === $id)
                                            {
                                                        $tmp_arr->$count =
                                                        $file;

                                                        $count++;
                                                        $found = TRUE;
                                            }
                                }

                                return ($found ? $tmp_arr : null);
                    }

                    return $files;

        }
        else
        {
                    return null;
        }
}

...

</ end of sample code>
```

8. As_questions-kirjasto

As_questions-kirjasto, nimensä mukaisesti huolehtii huutoihin liittyvien kysymysten ja kysymysten vastausten tallentamisesta/päivittämisestä ja niiden noutamisesta tietokannasta. As_questions-kirjastolla on esimerkiksi metodi, jolta voi pyytää yhteen huutoon liittyvät kysymys/vastaus sarjat. Alla oleva esimerkki kuvaa tätä metodia.

```
<?php

/**
./application/libraries/As_questions.php
*/

// ------------------------------------------------------------

/**
* get_questions
*
* get_questions-method in As_questions-class located at
/application/libraries
* /As_questions.php. Returns all matching questions
* @access           public
* @param            int         id of item questions are related
* @return           array of questions
*/
function get_questions($id = null)
{
        if(isset($this->_questions)
        && $this->_questions != FALSE)
        {
                    $questions = $this->_questions;
                    $tmp_arr = new stdClass();

                    $count = 0;
                    $found = FALSE;

                    foreach($questions as $question)
                    {
                                if($question->item_id === $id)
                                {
                                            $tmp_arr->$count = $question;
                                            $count++;
                                            $found = TRUE;
                                }
                    }

                    return ($found ? $tmp_arr : null);
                    }
```

```
        else
        {
                    return null;
        }
  }

...

</ end of sample code>
```

9. As_categories-kirjasto

As_categories huolehtii kategorioiden tallentamisesta, päivittämisestä ja noutamisesta tietokannasta ja kategoriapuun rakentamisesta näkymiä varten. Esimerkiksi kun haetaan tietokannasta yksi huuto, ei tämä yksi huuto tiedä vielä omasta kategoriastaan muuta kuin numeerisen arvon, joka on viittaus kategorioiden tietomalliin. Identifioidakseen oman kategoriansa, voi huuto pyytää esimerkiksi kategorian nimeä As_categories-kirjastolta. Alla oleva esimerkki kuvaa tätä.

```
<?php

/**
./application/libraries/As_categories.php
*/

// Kutsutaan As_categories-kirjaston metodia get ja annetaan
// parametreinä
// attribuutin nimi, kategorian id joka halutaan
$this->as_categories->get('name', 22);

...

</ end of sample code>
```

10. As_auctions-kirjasto

As_auctions-kirjasto huolehtii huudoista. Koska itse tuotteen huutamiseen liittyy paljon monimutkaista logiikkaa, luotiin tätä toiminallisuutta varten oma kirjasto. Samalla kun As_auctions-kirjasto huolehtii korotus logiikasta, pitää se sisällään myös rajapinnan huutojen tietomalliin, joten muu sovellus saa huutoihin liittyvän tiedon As_auctions-kirjastolta.

11. As_users-kirjasto

As_users-kirjasto, toisin kuin As_user-kirjasto, hallinnoi yhden käyttäjän sijaan kaikkia käyttäjiä. Koska huutoihin voi liittyy paljon käyttäjiä, tarvittiin tätä varten oma kirjasto. Huudolla voi olla kysymyksiä ja huutoja, jotka pitää voida identifioida johonkin käyttäjään kuuluvaksi, jolloin sovelluksen pitää tietää kuka esimerkiksi omistaa X huudon tai X huutoon liittyvän korotuksen. Useat sovelluksen muut kirjastot ovat riippuvaisia tästä kirjastosta. Esimerkiksi jos yhden huudon näkymässä halutaan jokaisen kysymyksen esittäjä identifioida omalla nimellään, onnistuu se As_users-kirjaston avulla.

12. As_items-kirjasto

As_items-kirjasto, nimensä mukaisesti huolehtii huudon kohteina olevista tuotteista, niiden tallentamisesta, päivittämisestä ja noutamisesta. As_items-kirjasto on rajapinta tuotteiden tietomalliin ja pitää sisällään automatiikkaa, jolla se yrittää päätellä missä näkymässä käyttäjä milloinkin on. Tämä ominaisuus on hyödyllinen latausaikoja ja suorituskykyä ajatellen, mutta helposti menettää tarkoituksensa esimerkiksi silloin, kun yhden huudon näkymään tuotetaan listaus kymmenestä suosituimmasta kohteesta. Tässä esimerkkinä annetussa tapauksessa joudutaan tietokantaan tekemään uusi kutsu ja etu on menetetty.

13. As_user-kirjasto

As_user-kirjasto käsittelee huutokauppaa selaavaa käyttäjää. Tämä kirjasto tarkastaa käyttäjän oikeudet jokaisella sivulatauksella, kirjaa käyttäjän sisään pyydettäessä, päivittää käyttäjän tiedot tarvittaessa ja seuraa käyttäjän toimia huutokaupassa. Käyttäjä voi olla anonyymi tai kirjautunut.

14. As_emails-kirjasto

As_emails-kirjasto huolehtii erilaisten sähköpostien, kuten rekisteröitymisen vahvistamista pyytävän sähköpostin lähettämisestä. Kirjautunut käyttäjä voi muokata omia asetuksiaan, joista löytyy mm. vaihtoehto sähköpostien vastaanottamisesta silloin kun käyttäjän omistamassa huudossa tapahtuu korotus.

15. As_events-kirjasto

As_events-kirjasto kirjaa kaikki käyttäjää koskevat tapahtumat, kuten esimerkiksi käyttäjän omistamalle tuotteelle asetetaan uusi kysymys. Esimerkiksi huutokaupan perustaja voi käyttää As_events-kirjastoa esittämään kirjautumisen yhteydessä käyttäjälle kaikki häntä koskettavat tapahtumat.

16. As_autobid-kirjasto

Huutokauppa-alustassa on toiminnallisuus joka mahdollistaa korotusautomaation käytön uuden huudon asettamisen yhteydessä. As_autobid-kirjasto on tehty johtuen korotusautomaation sisältämästä monimutkaisesta logiikasta. Tämä ongelma ja se ratkaisu haluttiin ympäröidä omaan luokkaansa, omaksi toiminnokseen ja korotusautomaatista huolehtii As_autobid-kirjasto.

17. As_maintenance-kirjasto

As_maintenance-kirjasto toimii eräänlaisena alustan talonmiehenä. As_maintenance-kirjasto huolehtii huutojen lopettamisesta, tiedostojen poistamisesta ja lokien tyhjennyksestä. As_maintenance-kirjasto suorittaa ylläpitotehtävänsä n. 5 minuutin välein.

Huutokauppa-alustan toiminnallisuudet

Alustan määrittelemät roolit käyttäjille

1. Roolin “Deny Access” (sovelluksen määrittelemä nimi) kuvaus.

Roolilla “Deny Access” ei ole oikeutta kirjautua. Tämä rooli on luotu tapauksia varten, joissa huutokaupan ylläpitäjä/ylläpitäjät haluavat estää käyttäjän pääsyn huutokauppaan.

2. Roolin ”Admin” (sovelluksen määrittelemä nimi) kuvaus.

Roolilla “Admin” on ylläpitäjän oikeudet. Ainoastaan “Admin”-roolin omaavat käyttäjät voivat kirjautua sovelluksen hallintaan.

3. Roolin ”Viewer” (sovelluksen määrittelemä nimi) kuvaus.

Roolilla “Viewer” ei ole oikeutta osallistua aktiivisten tuotteiden huutamiseen. “Viewer” on rooli, joka asetetaan rekisteröityneelle käyttäjälle kunnes huutokaupan ylläpitäjä/ylläpitäjät hyväksyvät käyttäjän tiedot ja päivittävät tämän roolin.

4. Roolin ”Bidder” (sovelluksen määrittelemä nimi) kuvaus.

Roolilla “Bidder” on oikeus osallistua tuotteiden huutamiseen, asettaa aktiivisten tuotteiden huutoja seurantaan ja asettaa aktiiviselle tuotteelle kysymyksiä.

5. Roolin ”Auctioneer” (sovelluksen määrittelemä nimi) kuvaus.

Roolilla “Auctionneer” on oikeus asettaa omia tuotteita huudettavaksi, osallistua aktiivisten tuotteiden huutamiseen, asettaa aktiivisia tuotteita seurantaan ja asettaa aktiivisille tuotteille kysymyksiä.

Alustan toiminallisuudet

1. Aktiivisten tuotteiden selaaminen

Huudettavia tuotteita voi selata listauksena, joka kattaa järjestelmän kaikki huudettavat tuotteet tai kategorioittain, jolloin näytetään aktiiviset huudot selattavan kategorian osalta. Aktiivinen huuto on tuote jonka huuto aika ei ole päättynyt.

2. Yhden aktiivisen tuotteen selaaminen

Huudettavaa tuotetta voi tarkastella omassa näkymässään jolloin näkyvillä on myös muuta tietoa joka liittyy yhteen huudettavaan tuotteeseen.

3. Käyttäjäksi rekisteröityminen

Osallistuakseen aktiivisten tuotteiden huutamiseen ja niihin liittyviin toimiin, käyttäjän pitää rekisteröityä huutokaupan käyttäjäksi. Käyttäjältä vaaditaan etunimi, sukunimi, syntymäaika, katuosoite, maa, kaupunki, postinumero, puhelin, sähköposti, käyttäjänimi ja salasana. Kun käyttäjä on antanut tarvittavat tiedot ja painanut lähetä, järjestelmä generoi käyttäjälle osoitteen. Alustan tuottama uniikki osoite lähetetään käyttäjän antamaan sähköpostiin, jota klikkaamalla käyttäjän antamat tunnukset aktivoidaan ja käyttäjä voi kirjautua huutokauppaan.

4. Omien tietojen muokkaaminen

Rekisteröitynyt ja kirjautunut käyttäjä voi muokata aikaisemmin tallentamiaan tietoja kirjautumalla rekisteröitymisen yhteydessä antamallaan salasanalla ja sähköpostiosoitteella huutokauppaan.

5. Omien asetuksien muokkaaminen

Rekisteröitynyt ja kirjautunut käyttäjä voi muokata huutokaupan asetuksia. Omissa asetuksissaan käyttäjä voi valita lähettääkö järjestelmä hänelle sähköpostia seuraavissa tilanteissa (käyttäjän pitää olla joko tuotteen omistaja, tuotteessa huutajana tai seuraa huutoa): kohteelle asetetaan uusi huuto, korotukseni ylitetään, kohteelle asetetaan uusi kysymys ja kohteen huuto päättyy.

6. Unohdetun salasanan palautus

Jos rekisteröitynyt käyttäjä unohtaa salasanansa, voi uuden tilata alustalta, jolloin se lähetetään käyttäjän rekisteröitymisen yhteydessä antamaan sähköpostiosoitteeseen.

7. Tuotteen asetus huudettavaksi

Kirjautunut käyttäjä joka omaa roolin “Auctioneer”, voi asettaa omia tuotteitaan huudettavaksi. Vaadittavia tietoja jotta huuto voidaan tallentaa on: myyntimäärä, nimi, kategoria, kuvaus, kunto, myyntiaika, maa, kaupunki, postinumero, katuosoite, maksutapa, toimitustapa, lähtöhinta, hintavaraus ja minimi korotus.

Kun vaadittavat kentät on täytettä ja käyttäjä tallentaa tuotteen, tuote jää odottamaan huutokaupan ylläpitäjän/ylläpitäjien hyväksyntään. Huutokaupan ylläpitäjä/ylläpitäjät hyväksyvät tuotteen tiedot ja aktivoivat tuotteen, jolloin kyseinen tuote näkyy huutokaupan listauksissa niin sanottuna aktiivisena huutona.

8. Omien aktiivisten huutojen selaaminen

Kirjautuneella käyttäjällä on oma näkymä, jossa voi selata tuotteita joissa on itse tuotteen omistajana.

9. Omien ei-aktiivisten huutojen selaaminen

Kirjautuneella käyttäjällä, jonka rooli on “Auctioneer” on oma näkymä, jossa voi selata tuotteita joiden huuto aika on päättynyt ja joissa käyttäjä on ollut omistajana.

10. Ei-aktiivisten tuotteiden joissa huutajana selaaminen

Kirjautuneella käyttäjällä, jonka rooli on joko “Auctioneer” tai “Bidder”, on oma näkymä, jossa voi selata tuotteita joiden huuto aika on päättynyt ja joissa käyttäjä on ollut huutajana.

11. Aktiivisten tuotteiden joissa huutajana selaaminen

Kirjautuneella käyttäjällä jonka rooli on joko “Auctioneer” tai “Bidder” on oma näkymä jossa voi selata tuotteita jotka ovat aktiivisia ja joissa käyttäjä on huutajana.

12. Aktiivisen tuotteen huutamiseen osallistuminen

Kirjautunut käyttäjä jonka rooli on joko “Auctioneer” tai “Bidder”, voi osallistua tuotteen huutamiseen. Antamalla summan ja tallentamalla antamansa summan tuotteen näkymässä käyttäjä osallistuu huutamiseen.

13. Aktiiviselle tuotteelle korotusautomaatin asetus

Kirjautunut käyttäjä jonka rooli on joko “Auctioneer” tai “Bidder”, voi automatisoida huutojen korotukset antamallaan korotusrajalla. Huutamisen yhteydessä käyttäjä voi aktivoida korotusautomaatin. Aktivoimalla “autobid” (sovelluksen määrittelemä nimi) kentän ja antamalla limit-kenttään (sovelluksen määrittelemä nimi) korotuskaton ja tallentamalla huudon. Aina kun tuotteelle tehdään uusi korotus toisen käyttäjän toimesta, sovellus asettaa korotuksen myös niiltä käyttäjiltä, jotka ovat samalle tuotteelle aktivoineet korotusautomaatin ja joiden korotuskatto on suurempi kuin tuotteen sen hetkinen korkein huuto.

14. Aktiivisen tuotteen huutojen seuraaminen

Kirjautunut käyttäjä jonka rooli on joko “Auctioneer” tai “Bidder”, voi asettaa tuotteen huudot seurantaan, jolloin tuotteen huutojen tilaa voi seurata omasta näkymästään.

15. Aktiiviselle tuotteelle kysymyksen asettaminen

Kirjautunut käyttäjä jonka rooli on joko “Auctioneer” tai “Bidder” ja joka ei ole tuotteen omistaja, voi asettaa tuotteelle kysymyksen.

16. Aktiiviselle tuotteelle asetettuun kysymykseen vastaaminen

Kirjautunut käyttäjä jonka omistamalle tuotteelle on asetettu kysymys, voi vastata asetettuun kysymykseen.